In compliance with the
Canadian Privacy Legislation
some supporting forms
may have been removed from
this dissertation.

While these forms may be included
in the document page count,
their removal does not represent
any loss of content from the dissertation.

University of Alberta

Model Based Approach to Agent Communication and Defining Communication Error Conditions

by

Anita Petrinjak

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of ***Master of Science***

Department of Computing Science

Edmonton, Alberta
Fall 2003

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisisitons et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-87861-9
Our file Notre référence
ISBN: 0-612-87861-9

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou aturement reproduits sans son autorisation.

Canada

University of Alberta

Library Release Form

Name of Author: *Anita Petrinjak*

Title of Thesis: *Model Based Approach to Agent Communication and Defining Communication Error Conditions*

Degree: *Master of Science*

Year this Degree Granted: *2003*

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Date: July 9, 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Model Based Approach to Agent Communication and Defining Communication Error Conditions** submitted by **Anita Petrinjak** in partial fulfillment of the requirements for the degree of **Master of Science**.

Date: June 20, 2003

Abstract

Communication is essential part of cooperative activities in multi-agent systems. This research defines an agent interaction model, which is used as the basis for the agent communication, and from which error conditions for agent communication can be defined - cases in which an agent generates a *not-understood* message. The model specifies task and agent interdependencies, agent roles, and predicate properties at a domain-independent level of abstraction. It also defines which agent beliefs may be updated, revised, or accessed through a communication act from another agent in a particular role. The model is represented in a declarative form using XML. A simple multi-agent system was developed in which agents use the underlying interaction model to detect deviations from normative communication behavior and to constrain the allowable belief updates that might occur within a *not-understood* dialogue.

Acknowledgements

I want to thank my supervisor Dr. Renee Elio for her ideas and guidance, and for all the invaluable support and help.

I wish to thank the members of my examining committee, Dr. Michael Carbonaro and Dr. Eleni Stroulia, for their valuable input.

I would like to thank my friends and colleagues for support and encouragement: Yaser Khamayseh, Paul Ferry, Tracie Scott, Cindy Want, Maria Cutumisu and Akihiro Kishimoto. I would also like to thank my friends Ivana Pešić Jepsen and Tanja Bosak for being there for me even though they were far.

Many thanks to my parents Ruža and Đuro Petrinjak, and to my sister Maja Petrinjak for their support and love.

Table of Contents

CHAPTER 1 INTRODUCTION 1

1.1 MOTIVATIONS AND OBJECTIVES 1
1.2 THESIS ORGANIZATION OVERVIEW 5

CHAPTER 2 LITERATURE REVIEW 7

2.1 SPEECH ACT THEORY AND THE BDI PERSPECTIVE 7
2.1.1 SPEECH ACT THEORY 7
2.1.2 THE BDI PERSPECTIVE AND INTENTION THEORY 10
2.2 AGENT COMMUNICATION LANGUAGES 13
2.2.1 DARPA KNOWLEDGE SHARING EFFORT AND KQML 14
2.2.2 FIPA ACL STANDARDIZATION EFFORT 17
2.2.3 FROM MESSAGES TO PROTOCOLS TO CONVERSATION POLICIES 20
2.2.4 SUMMARY 22
2.3 COMMUNICATION AND COOPERATION 24
2.3.1 JOINT INTENTION AND JOINT RESPONSIBILITY 24
2.3.2 TASK BASED DIALOGUE 25
2.3.2.1 Plan-based Speech Act Theory 26
2.3.2.2 Theoretical constructs for task related dialogues: SharedPlan 28
2.3.2.3 Plan based models of dialogue 32
2.3.2.4 The Knowledge Preconditions 33
2.3.2.5 Collagen 34
2.3.2.6 The Abstract Task Model 38
2.3.2.7 Summary 39
2.3.3 TASK MODELING AND COMMUNICATION 39
2.3.3.1 TAEMS framework 40
2.3.4 MULTI-AGENT FRAMEWORKS 45
2.3.4.1 InfoSleuth and Retsina 46
2.3.4.2 Comparison between the (AI) Multi-agent Systems and the Task Based Dialogue Frameworks 47
2.4 DISCUSSION 48

CHAPTER 3 AGENTS AND INTERACTION MODEL 51

3.1 A MOTIVATING EXAMPLE: DISTRIBUTED AIRLINE TRAVEL PLANNING PROBLEM 53

3.2 What is an agent? 55
3.2.1 Agent Functionality: Problem Solving 55
3.2.2 Agent Functionality: Communication plans 56
3.3 Modeling Task and Agent Interdependencies 57
3.3.1 Predicate and Predicate classes 59
3.3.2 Axioms and Inference Rules for Agent reasoning 63
3.4 Assimilating Speech Acts: Occasions for not-understood 65
3.5 Summary 70

CHAPTER 4 INTERACTION MODEL AND MULTI-AGENT SYSTEM IMPLEMENTATION ..72

4.1 Agent Interaction Model Implementation 73
4.1.1 Task structure ontology 73
4.1.2 Explicit Communication Object Ontology 77
4.1.3 Interaction Model Synthesis Form 79
4.2 Agent Architecture and Multi-Agent System Implementation 83
4.2.1 Multi-agent system 83
4.2.2 Agent and Agent Execution Cycle 83
4.2.3 Task Operators 85
4.2.4 Communication Plans 89
4.2.5 Task Domain Abilities 92
4.3. Summary 92

CHAPTER 5 SYSTEM BEHAVIOR 94

5.1 Overview of the system behavior: Problem solving process and communication 94
5.2 Test cases of the regular behavior 98
5.3 Not-understood responses and conversations 102
5.3.1 Overview 102
5.3.2 On public vs. private predicates 104
5.3.3 On defeasible vs. fixed predicates 106
5.3.4 Is the illocutionary force applied to the predicate allowed? 106
5.3.5 Proper agent role to change a defeasible predicate via communication 108
5.3.6 Interaction model mismatch between agent roles and dependencies 109
5.4 Summary and Discussion 110

CHAPTER 6 DISCUSSION AND FUTURE DIRECTIONS 112

6.1 Conclusion and Discussion 112
6.2 Future directions: Agents and Software Engineering 114

BIBLIOGRAPHY 118

APPENDIX A: FIPA ACL SEMANTICS 125

APPENDIX B: DOCUMENT TYPE DEFINITION 126

APPENDIX C: SUPPORTING TASK OPERATORS 128

APPENDIX D: MESSAGE GENERATION ROUTINES 131

APPENDIX E: CONVERSATION TRACES 133

List of Tables

Table 2.1. Searle's conditions for *assert* and *question* illocutionary acts (S denotes sender, H denotes hearer) 10
Table 3.1 Illustrative belief state elements for agent j (from Figure 3.2) 59
Table 3.2 A set of the predicate classes 61
Table 3.3 Internal bookkeeping beliefs 62
Table 3.4 The schema for the *inform* message type and six categories of *not-understood* 66
Table 3.5 The schema for the *disconfirm* message tupe and six categories of *not-understood* 68
Table 3.6 The schema for the query and six categories of *not-understood* 70
Table 4.1 *Task*, *Constraint* and *Variable* classes as part of the task structure ontology 74
Table 4.2 *Method* class as part of the task structure ontology 76
Table 4.3 Abstract task operators (in pseudo code) 87
Table 4.4 Supporting task operator S1 (in pseudo code) 88
Table 4.5 Message filtering and checking plans 91
Table 5.1 Test scenarios 98

List of Figures

Figure 2.1 Agent communication layers: 'Message' denotes communication act from an ACL; 'Communication' refers to the means by which a message is transported from one agent to another; 'Content' is the content of the communication act (from Patil et al. (1997)) 14
Figure 2.2 An example of the state transition conversation protocol (the basic conversation for action) (from Winograd and Flores (1986)) 20
Figure 2.3 REQUEST and INFORM plans 27
Figure 2.4 A discourse structure illustration 30
Figure 2.5 Collagen: Internal discourse state representation (from Rich and Sidner (1998)) 36
Figure 2.6 Collagen test application screen (from Rich and Sidner (1998)) 37
Figure 2.7 A representation of a simple TAEMS task model (from Wagner et al. (1999)) 41
Figure 2.8 A TAEMS conversation protocol example (from Wagner et al. (1999)) 45
Figure 3.1 Task decomposition of the airline travel planning problem 53
Figure 3.2 Elements of task structure and derived agent roles 58
Figure 4.1 Agent architecture diagram 73
Figure 4.2 Excerpt from a task structure in XML 77
Figure 4.3 Explicit Communication Ontology in XML 79
Figure 4.4 A context sensitive grammar for the legal *inform* and *disconfirm* messages allowable by an interaction model (*s-to-r = sender-to-receiver*) 80
Figure 4.5 A context sensitive grammar for the legal *query-ref* messages allowable by an interaction model (*s-to-r = sender-to-receiver*) 82
Figure 4.6 Multi-agent system execution cycle 83
Figure 4.7 Agent execution cycle 84
Figure 4.8 Task state transition diagram with abstract task status states 85
Figure 5.1 A test task structure 96
Figure 5.2 A simple test task 99
Figure 5.3 A simple test task 99

Chapter 1 Introduction

1.1 Motivations and Objectives

Communication is essential part of cooperative activities. Cooperative behavior can occur between humans, software agents, or it can include both humans and agents. If cooperative entities are involved in relatively complex, joint problem solving, communication plays crucial role in the coordination of problem solvers' activities and in enabling task advancement through knowledge exchange. Different agents may be responsible for different parts of a task, may have different problem solving abilities, and access to different resources. Since agents may be quite diverse in their knowledge, reasoning and other abilities, the question 'how could they communicate' is posed. This question motivated the development and standardization of high-level agent communication languages (ACLs) independent of agent implementation. The Federation for Intelligent Physical Agents (FIPA) defines one such standard for an agent communication language (FIPA, 2003), providing a set of core message types (e.g., *inform*, *request*, *query*). For example, the form of an *inform* message is *(inform :sender agent-i :receiver agent-j :content weather (today, raining) :conversation-id 001)*, where *i* and *j* are agents, and *i* informs *j* that proposition *c* is true. Another ACL specification was proposed by the DARPA Knowledge Sharing Effort (Patil et al., 1997) in the form of KQML (Labrou and Finin, 1994; Labrou and Finin, 1997), also giving a set of core message types (e.g. *tell*, *ask-if*). The work on agent communication languages borrowed heavily from earlier research in philosophy and linguistics, most notably Bratman's theory of intention (Bratman, 1990) and speech act theory (Austin, 1962; Searle, 1965, 1969). These other disciplines were instrumental in the evolution of the so-called 'belief-desire-intention' perspective of agency. Under this perspective, folk psychology notions like belief, desire, and intention were used to designate mental states in order to develop theories of rational action. Speech act theory was revolutionary in its characterization of utterances as actions upon a world and hence communication could be viewed in a traditional AI planning sense, i.e., a series of actions with pre and post conditions that are taken to achieve some desired world change. The resulting development of ACLs has given great attention to the general syntax of the core message types and in some cases (FIPA's ACL) formal semantics for each core message type, based on modal logics of belief, desire, and intention.

The significant effort in specifying the syntax and semantics for agent communication languages does not address the matter of what agents actually 'talk' *about*, i.e., what is the actual content (the *:content* field) of a message, even when there is a shared ontology and a content

language. Agent communication languages do not constrain meaning of a message. Consequently, plausible interpretations, i.e. intentions behind a message to be understood, are unconstrained, which poses a difficult problem to be resolved. Greaves et al. (2000) vocalized the concerns of the agent communication community due to the complexity of the above problem. They argued that richness and expressiveness of ACLs could prevent agents from understanding the meaning of messages due to many-to-many mapping between speech acts and communicative goals (intentions). Elio and Haddadi (2000b) proposed that what cooperating or coordinating agents talk about is their problem-solving progress at some abstract level and this abstract specification of task progress serves as part of a jointly held conversation policy (Greaves et al., 2000).

Our research has adopted a pragmatic approach towards specifying foundations for constraining and understanding the meaning of a message in the form of an agent *interaction model*. This model defines static and dynamic properties of agent relationships and roles, and task properties and interdependencies. In essence, our aim is to provide a context definition in a structured manner to enable agent communication. The thesis of this research is that the components of an agent interaction model (as we define it) play the role of context whose function is to define the meaning of a message in such a way as to enable interpreting messages dynamically during problem solving process. We evaluate this position by showing how a multi-agent system which uses an interaction model can successfully coordinate and advance its task by means of communication. Moreover, evaluation can be provided by looking at the 'other side' of understanding the message, i.e. by defining what it means for a message to be *not understood* via an ontology of message error conditions. We assume that agents will send only meaningful messages dependant on their beliefs about shared task and agent roles. Therefore, we evaluate the robustness of the interaction model by providing agents with inconsistent beliefs about their task relations and their roles, observing how they deal with the messages that they do not understood, how and if they overcome inconsistencies in their knowledge, and continue with the problem solving process.

FIPA includes a *not-understood* message as one of its core message types. The importance of this message type is implicitly expressed in the fact that the first requirement imposed on a FIPA ACL speaking agent is that it has to be able to generate at least one message, namely *not-understood*. The form of this message is *(not-understood :sender i :receiver j :content c)*, where *i* and *j* are agents, and *not-understood* message is i's response to j in the context of some previous message from *j* to *i*. The message content *c* is defined as a tuple consisting of an action

or event (e.g., a just-received inform message from *j* to *i*) and a 'reason'. The occasion for sending *not-understood* is described as follows:

> The sender of the *not-understood* communicative act received a communicative act that it did not understand. There may be several reasons for this: agent may not have been designed to process a certain act or class of acts, or it may have been expecting a different message. For example, it may have been strictly following a pre-defined protocol, in which the possible message sequences are predetermined. The *not-understood* message indicates to the sender of the original (misunderstood) action that nothing has been done as a result of the message.... The second element of the message content tuple is a proposition representing the reason for the failure to understand. There is no guarantee that the reason is represented in a way that the receiving agent will understand. However, a cooperative agent will attempt to explain the misunderstanding constructively. (FIPA, 2003)

The work we describe here proposes the interaction model as the underlying structure that defines the meaning of messages. We proceed further to show that the interaction model can also be used for defining and structuring reasons for generating a *not-understood* message type, and also for defining possible responses to receiving a *not-understood* message. A 'constructive explanation' for a *not-understood* can be specified as a deviation from this underlying interaction model that defines expected, legal, and pragmatic messages in a more general way than a predefined conversation protocol. The premise is that message understanding is grounded in the extended perlocutionary effects of that message act, i.e., how a message changes an agent's informational state in such a way that the agent behaves differently in the context of some task environment.

By FIPA's formal semantics, if agent *i* informs agent *j* of some proposition α, agent *i*'s intended effect is that *j* adopt belief in α. Agent *j* may not do that for a number of reasons, and therefore the intended effect is just that—intended but not guaranteed. In making a communication act, a speaker is trying to change an aspect of a non-deterministic world that is not directly accessible, namely the mental state of the hearer or receiving agent. Hence, *not-understood* is quite a pointed response back to agent *i* from that inaccessible world, indicating that the action was not just unsuccessful but, to put it crudely, dead on arrival for some other reason. We propose an ontology of error conditions in agent communication based on the

conditions that must be met before *j* would even attempt to assimilate α into its informational state.

For example, suppose that agents A1 and A2 are cooperating on some joint task that involves a database of movie information. Suppose that A1 informs agent A2 that "movie database has crashed". The issue at hand is how A2 can reach the correct interpretation of the message, which in turn depends on recognizing the A1's intention behind the message. In what sense do we wish to say that A2 has 'understood' this *inform* message? At the very least, A1 wished A2 to adopt belief in the proposition "the movie database has crashed." This is, in fact, what FIPA specifies as the intended effect of an *inform* message type. But we can imagine cases in which A2 does *not* understand this message, i.e., does not even assimilate the proposition into its belief state. Perhaps it does not know the term "crash." Perhaps it cannot account for why agent A1 would even have knowledge about the state of the movie database (e.g., it believed A1's responsibilities concerned the airline database instead). Perhaps knowledge of the state of the movie database is totally irrelevant to A2's concerns, and it cannot account for why A1 would be sending this *inform* in the first place. Simply put, it is entirely possible for A2 to adopt belief in the fact that "the movie database has crashed" but this might not have any impact on A2's behavior. Does agent A1 think that it might? This speaks to the pragmatic conditions that A2 thinks are, or are not, behind A1's inform message in the first place. In general, the models that these two agents have of each other and each other's tasks serve as a context for resolving the intention of a message, and for defining what it means for a message to be 'understood' or 'not understood'. The interaction model is this overarching context in which agents attempt to resolve incoming messages.

Beyond being used just as an indirect evaluation method of interaction model, we believe that looking into *not-understood* message type has theoretical and pragmatic value in itself. First, at the most general level, the *not-understood* issue frames message understanding in terms of the extended perlocutionary effects of a communication act on the behavior of an agent. For example, Pitt & Mamdani (2000) present protocol-level semantics that include an 'add function'—an agent's procedure for computing the change in its information state from the content of an incoming message delivered in the context of a particular protocol. The work we present here can be viewed as a way to define certain elements of that add-function. Second, if we seriously regard the notion of agent as a kind of programming abstraction (Shoham, 1993), then *not-understood* can be seen as signaling a 'run-time error' when the effect of a communication action from one agent to another would constitute an illegal or impossible action on the receiving agent's internal (mental) state.

Bearing in mind that an interaction model should be common knowledge among agents about their responsibilities, roles, tasks, and interdependencies, we define a representational scheme for shared knowledge in the software engineering standard XML/DTD. We consider some aspects of the interaction model developed here to be similar to the methodologies being proposed to cross the gap between agent paradigm and software engineering paradigms (Jennings, 2000). We also believe that coupling software engineering technology with high-level agent communication would be beneficial for both sides. In addition to gaining an adequate foundation for the conversations and problem solving in general, agents can extend their applicability scope by using standardized and widely used software technologies. Software engineering systems can utilize high-level agent communication as the powerful and flexible means of knowledge exchange.

In summary, we use the term interaction model to define what agents can, must, or might talk about during their joint problem solving, and how this can be further constrained through the specification of agent roles. A *not-understood* is the detection of an inconsistency between what is expected by that model and what is observed. The 'constructive reason' is the specific model violation, stated at the same abstract level as the shared model. It would be quite useful if receipt of a *not-understood* message changed agent *i*'s internal state, in such way that *i* would not resend the original (misunderstood) message back to *j*. The specification of a reason leaves open the possibility for belief revision on either agent's part, and this is a complicated matter. We give some preliminary ideas on how an interaction model might specify policies about this issue as well.

1.2 Thesis Organization Overview

The work in this thesis is situated within agent communication and cooperation research area. We investigate the possible components of an agent conversation policy in the form of agent interaction model, covering the issue of a *not-understood* message type in principled manner.

Chapter 2 discusses the key theoretical elements upon which this work is based, namely speech act theory, theory of agency and intention, agent communication languages, and the notion of conversation protocols. It also discusses closely related research on task-directed dialogue from both computational linguistics and agent community perspectives. Chapter 3 presents the elements of the interaction model in more detail, and relates it to the similar work in the field. Chapter 4 describes the implemented multi-agent system and agent architecture which uses the interaction model for cooperative problem solving on a simple airline reservation task. Chapter 5

presents the behavior of the system, the results it produced and the conversations it generated while dealing with different problem solving situations. The final chapter discusses the work to date, its relation to software engineering paradigms, its limitations, and directions for future work.

Chapter 2 Literature Review

Software agent communication and cooperation are complex issues that have received lots of attention from the research circles. However, there are two key theoretical foundations that have influenced agent theory and practice. The first is the so-called *belief-desire-intention* (BDI) perspective of rational agency (Bratman et al., 1988; Rao and Georgeff, 1995; Georgeff et al., 1999). This notion traces its routes to Bratman's theory of intention (1990). The second is the speech act theory, developed by Austin (1962) and Searle (1965; 1969) within the linguistic community. The elements of both of these perspectives are central to the multi-agent theory and practice, and this chapter will begin by reviewing them.

Following upon these theories, we review the two specifications that have emerged for agent communication languages (ACLs), namely KQML and FIPA ACL. KQML was born out of DARPA's Knowledge Sharing effort (Patil et al., 1997), and FIPA ACL standard was proposed by FIPA (FIPA, 2003), an organization for software agent related standards. The agent communication languages are heavily grounded in the above mentioned theoretical notions, namely BDI agency and speech act theory.

What is central, however, to this thesis is the role that task knowledge plays in how agents structure and interpret their messages in the context of cooperative problem solving and complex interactions. Therefore, the final area this chapter will consider in some detail is the matter of task modeling and task directed dialogue, from both AI agent community perspective as well as from the discourse processing community.

2.1 Speech Act Theory and the BDI Perspective

2.1.1 Speech Act Theory

Speech act theory and the belief-desire-intention model of agency are related through the key concept of intention. The importance of the speech act theory is that it introduced the idea that utterances are the actions upon the world. This also means that utterances, just as any other actions, have underlying intentions and can be planned. Austin (1962) presented the utterances as *speech acts* or *performatives* (action performing statements) in his seminal work for the speech act theory. His examples vividly describe the idea:

(E. a) 'I do (sc. take this woman to be my lawful wedded wife)' - as uttered in the course of the marriage ceremony.

(E. b) 'I name this ship the *Queen Elizabeth*' - as uttered when smashing the bottle against the stem.

(E. c) 'I give and bequeath my watch to my brother' – as occurring in a will.

(E. d) 'I bet you sixpence it will rain tomorrow.' (Austin, 1962, p.5)

Each of the utterances in the above example does not simply state or describe things, actions, etc. They perform the actual action whether it is the action of getting married, naming the ship, giving the watch to one's heir or betting. Simply put, these utterances cause some thing in the world to be true.

As the argument unfolds, Austin presents the idea that each speech act has three aspects of 'acting' associated with it: the locutionary force, the illocutionary force, and the perlocutionary force. The locutionary force consists of the sound of uttering certain noises, of the particular vocabulary, and of the 'meaning' and reference. The illocutionary force is the (conventional) force of a speech act that performs informing, betting, promising, warning, ordering, etc. Importantly, it denotes the speaker's intention behind a speech act. The perlocutionary force is what is achieved by saying something, the consequence or the effect on the audience such as surprise, persuasion, annoyance and knowing. Let us clarify this with Austin's example (1962, p.101-102): "Shoot her"

Act (A) or Locution

He said to me 'Shoot her!' meaning by 'shoot' shoot and referring by 'her' to her.

Act (B) or Illocution

He urged (or advised, ordered, &c.) me to shoot her.

Act (C. a) or Perlocution

He persuaded me to shoot her.

Act (C. b)

He got me to (or made me, &c.) shoot her.

Act B carries the intention of the speaker to urge the hearer "to shoot her". It can be observed that Act C (both *a* and *b* parts) presents the consequence of Act B, i.e. speaker's urging the hearer to shoot her resulted in the hearer being persuaded to do it. This structure holds two

levels of intentions. The first and more obvious level of intentions is reflected in the illocutionary force. In our example, it is the straightforward intention to urge the hearer to shoot somebody. The second level of intentions is related to the perlocutionary force, we can also call it the *internal* intention. In the above example, the speaker's (internal) intention to persuade the hearer to shoot somebody led him to adopt the intention to urge the hearer (to shoot her).

In his speech act theory, Searle (1969) looks into the issue of intentions underlying the speech acts. He presents a more detailed account of the illocutionary force and analyzes conditions related to the performance of speech acts. As a result, Searle proposes the set of conditions that are necessary and sufficient for the success of the speech act. They can be grouped in the following categories: the *input/output* conditions, the *propositional content* conditions, the *preparatory* conditions, the *sincerity* conditions, and the *essential* conditions. These conditions have been used as the basis for the semantics of the first fully-fledged agent communication language – KQML – and therefore we will examine them more closely.

The input/output conditions state that there has to be an 'open communication channel' between the speaker and the hearer. The speaker can utter (or send) a meaningful message, and the hearer can hear it (or receive it), and understand it (this condition excludes cases such as deafness and telling jokes). The propositional content conditions impose grammatical and conceptual restrictions on the propositions that can be included as the content of the speech acts. The preparatory conditions express what *has to be true* for the speaker in order to be able to carry out the speech act, e.g. the speaker has to have the knowledge of the proposition it intends to assert. The sincerity condition stipulates the idea of the 'sincere' speaker, e.g. speaker has to intend to do what s/he promised to do or has to believe in what s/he asserts. The essential condition relates to the speaker's intention to generate the speech act consequence, i.e. it denotes what speaker really is trying to achieve. For example, the consequence of the promising act is to make the hearer believe that the speaker is under obligation to do the promised act; the consequence of asking is to make the hearer give the answer, and so forth. The way to satisfy the essential condition is, according to Searle, to make the hearer understand this intention to produce a certain effect.

Type of condition	Type of act, Example	
	Assert, state (that), affirm *(I tell you that) the sun is shining.*	Question *(I ask you) is the sun shining?*
Propositional content	Any proposition *p*.	Any proposition or propositional function.
Preparatory	1. S has evidence (reason, etc.) for the truth of *p*. 2. It is not obvious to both S and H that H knows (does not need to be reminded of, etc.) *p*.	1. S does not know 'the answer', i.e., does not know if the proposition is true, or, in the case of the propositional function, does not know the information needed to complete the proposition truly.
Sincerity	S believes *p*.	S wants this information.
Essential	Counts as a declaration that *p* represents an actual state of affairs.	Counts as an attempt to elicit this information from H.

Table 2.1. Searle's conditions for *assert* and *question* illocutionary acts (S denotes sender, H denotes hearer)

Table 2.1, adapted from Searle (1969, p.66-67), shows examples of necessary and sufficient conditions for the *assert* (*state that, affirm*) and the *question* illocutionary acts. The next section investigates intentions more closely.

2.1.2 The BDI Perspective and Intention Theory

The belief-desire-intention (BDI) notion is at the core of the rational agency. Beliefs, desires and intentions are a set of attitudes and the so-called *pro-attitudes*, i.e. the mental states that guide agent's actions and have their roots in *folk psychology* (Wooldridge and Jennings,

1995). Beliefs represent the agent's knowledge about the world and this knowledge may be imperfect, as Georgeff et al. (1999) elaborate. Desire represents the desired end state that rational agent may try to achieve if it commits to it. This brings us to intentions that can be described as agent's commitment to fulfill its desires, and that create the rational, goal oriented behavior in agents.

The origins of the BDI perspective can be found in the intention theory. Bratman (1990) established the intention theory (as we know it) by resolving such issues as how to relate intentions to beliefs and desires, and to practical reasoning. Furthermore, he tackled the issue of the temporal stability of intentions.

Bratman (1990) argues that intentions are the fundamentals of the rational behavior for us as agents with limited resources. He states that intentions direct the way we act in the future, and calls them *future-directed intentions*, as explained further below:

> Why bother with future-directed intentions anyway?... I think there are two main answers. The first is that we are not frictionless deliberators. Deliberation is a process that takes time and uses other resources; this means that there are obvious limits to the extent of deliberation at the time of action. By settling on future-directed intentions, we allow present deliberation to shape later conduct, thereby extending the influence of deliberation and reason on our lives. Second, and relatedly, we have pressing needs for coordination. To achieve complex goals, I must coordinate my present and future activities. And I need also to coordinate my activities with yours. Future-directed intentions help facilitate both intra- and interpersonal coordination. (Bratman, 1990)

In order to make the connection between intentions and future actions explicit, Bratman calls intentions "conduct-controlling pro-attitudes" that pose problems to be solved and lead an agent into making plans for the future. For example, if a person has an intention to go to Calgary tomorrow, he needs soon to resolve how to get there, that is, to fill in the plan with specifications how to do it. The important property of intentions is that they serve as a filtering mechanism for adopting other intentions since intentions should be consistent. Such principle makes reasoning about possible intentions tractable for resource-bounded reasoners. For example, a person with the intention to go to Monterey tomorrow also has the intention to leave the car to his wife, and therefore he cannot consistently adopt intention to go to Calgary with the car.

His argument extends further into the discussion that if we are to have prior intentions and make future plans, our intentions and plans need to have a certain level of stability. Bratman proposes to have intentions considered stable by default, and to reconsider them only in special situations. For example, it does not help a person to constantly reevaluate whether to go to Calgary by plain or by bus since he cannot continue planning his trip (e.g. planning to buy a ticket) without settling on one choice. On the other hand, if he intends to go by bus, and he learns that bus drivers are on strike tomorrow, it would be rational of him to reevaluate his decision.

In relation to the beliefs, intentions should be consistent with one's belief set, e.g. a person cannot intend to go to the museum with his/her car and have a belief such as 'I do not have a car'. In contrast, the desires can be inconsistent, e.g. a person can desire to go to the museum next Friday, but that person can also desire to go to the theater next Friday. This follows from the notion that one's desires are competing to become intentions, and once a person has some intentions, other desires are filtered into the set of intentions through the consistency check with the existing intentions. If we use the previous example, a person who intends to go to the museum next Friday cannot adopt his/her desire to go to the theater at the same time as an intention.

Bratman's intention theory established the key principles and relations among intentions, beliefs and desires. His work was further developed into two main areas. The first one is the BDI agent approach (including BDI agent architectures). The full account of the BDI agent theory is out of the scope for this thesis, but can be further explored in, for example, Rao and Georgeff (1995), Georgeff et al. (1999). Examples of the BDI agent architecture can be found in Bratman et al. (1988) and Ingrand et al. (1992). The second direction is the development of the formal semantics for the belief, desire and intention notions. Wooldridge and Jennings (1995) give an overview of different formal semantics that were developed with intention to capture Bratman's theory key elements.

Cohen and Levesque (1990) developed one of the more notable examples of the intention theory formal semantics. One of the ideas behind the theory says that intentions are the crucial mechanisms for the resource-bounded agents; when agent commits to having some intentions, it can stop evaluating its competing desires and preserve its limited resources. In Cohen and Levesque's semantics, intention is defined through the notion of the *persistent goal*. A persistent goal reflects very committed attitude: an agent is trying to achieve a goal until it believes it has been satisfied or until it believes that the goal will never be true. Thus, the intention to do an action A is a persistent goal to have done action A. In Cohen and Levesque's words, "it seems reasonable to require that the agent be committed to believing he is about to do the intending action, and then doing it". Additionally, they also propose a second type of intention – 'intend-

that'. This is a commitment to bring about a particular world state (in which some proposition *p* is true). It is defined as agent's commitment to do a sequence of actions that will bring about this world state (in which this proposition *p* is true), although the agent may not have the complete knowledge of the required sequence of actions.

Cohen and Levesque also introduced the concept of *relativized persistent goal* by extending the circumstances under which the goal can be dropped. They present additional *reason* for dropping a goal (consisting of certain beliefs not holding any more). For example, "x could adopt the persistent goal to buy an umbrella relative to his belief that it will rain. He could then consider dropping his persistent goal should he come to believe that the forecast has changed" (Cohen and Levesque, 1990).

In summary, this section reviews the notion of intention from different perspectives, namely the speech act theory and the BDI agency theory. Belief, desire and intention are (originally) folk psychology constructs that have proved to be very useful notions as part of the rational agency theory. The speech act theory assumes rational and goal-oriented agent and it is, therefore, strongly relying on the BDI perspective as the core of the rational agency. Austin and Searle's key contribution is the idea of the utterances as actions upon the world, i.e. upon the mental state of the hearer. A person can perform an action of closing the window, that person can also take an action to convince the hearer to close the window (through a speech act). But, the speaker can also perform an action by convincing someone that some proposition is true (through a speech act). The key to the speech act understanding is in recognizing this underlying intention.

2.2 Agent Communication Languages

Agent communication language (ACL) is a high-level language with a purpose to enable heterogeneous agents to exchange information, primarily about their tasks. It defines a set of the core message types (also called communication acts or performatives) with their associated meanings. The task of an ACL performative is to convey an explicit 'meaning' (or illocutionary force) (without indirect speech forms). Here is an example of a core message from one of the ACLs (FIPA ACL): *(inform :sender agent-i :receiver agent-j :content "weather(tomorrow, overcast)" :language Prolog)*.

ACL core messages present the independent layer ("Logic of Communication" layer) from the content layer and the communication layer as shown in Figure 2.1 (Patil et al., 1997). Although the content is a part of the ACL message, it is (mostly) ignored by the ACL. Similarly,

ACL is ignorant of the communication layer mechanisms. KR and KIF in Figure 2.1 mean Knowledge Representation and Knowledge Interchange Format, respectively.

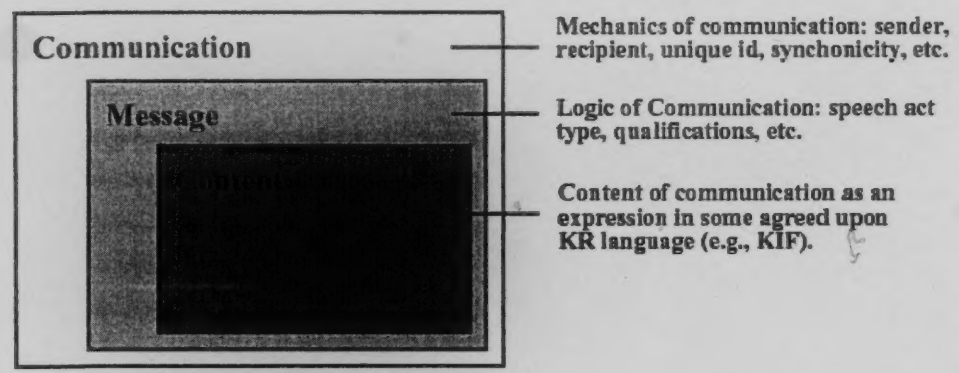

Communication
Message
Mechanics of communication: sender, recipient, unique id, synchonicity, etc.
Logic of Communication: speech act type, qualifications, etc.
Content of communication as an expression in some agreed upon KR language (e.g., KIF).

Figure 2.1 Agent communication layers: 'Message' denotes communication act from an ACL; 'Communication' refers to the means by which a message is transported from one agent to another; 'Content' is the content of the communication act (from Patil et al. (1997))

In a typical scenario, agents do not exchange only one message, but a sequence of messages, i.e. they engage in the conversations. The agent communication languages do not resolve the issue how to combine the messages into conversations. In the following sections we will talk about two ACL standards that have emerged, and we will extend our insight into the conversation protocols that structure message exchanges.

2.2.1 DARPA Knowledge Sharing effort and KQML

The Knowledge Sharing Effort was a project sponsored by DARPA and motivated by the lack of methodologies for sharing and communicating knowledge between software systems. According to Patil et al. (1997) the goal was "to develop a technology that will enable researchers to develop new systems by selecting components from library of reusable modules and assembling them together. ... In this way, declarative knowledge, problem solving techniques and reasoning services could all be shared among systems." As part of this general Knowledge Sharing Effort, the Knowledge Query and Manipulation Language (KQML) was conceived. Although it was not the initial idea, KQML started to be used as an agent communication language.

Initially, KQML existed only with the syntax specifications, and informal and partial semantics (Labrou et al., 1999). Since it was KQML's weakest point and an obstruction for using it, there were some efforts to supply semantic for KQML. We will look at the semantics proposed for KQML presented by Labrou and Finin (1994) and Labrou and Finin (1997) that is based on the Searle's analysis of the speech act, and his necessary and sufficient conditions for its success (as detailed above). This approach specifies preconditions, postconditions, and completion conditions for each performative. Some of the KQML performatives are *tell*, *deny*, *ask-if*, *error* and *eos* (end of stream). The following is the example of a KQML message:

```
(ask-if  :sender A
         :receiver      B
         :language      prolog
         :ontology      genealogy
         :reply-with    ID01
         :content       "mother(Suzan, Ann)" )
```

In the shown example, *ask-if* is the performative, and the rest are performative arguments organized in the keyword/value pair fashion. The keyword names are intuitive: they refer to the sender and receiver agents, they specify the ontology and the language used in the content expression (ontology and language parameters determine the syntax, semantics, and vocabulary of the expressions allowed in the content field.), and the last argument is the content expression itself. 'Reply-with' is used for keeping the track of answers to the sent messages.

Before we take a closer look at the KQML semantics, let us specify the framework for semantics in terms of the propositional attitudes (Labrou and Finin, 1997):

1. BEL, as in BEL(A,P), which has the meaning that P is (or can be proven) true for A. P is expression in the native language of agent A.
2. KNOW, as in KNOW(A, S), expresses knowledge for S, where S is a state description (the same holds for the following two operators).
3. WANT, as in WANT(A, S), to mean that agent A desires the cognitive state (or action) described by S, to occur in the future.
4. INT, as in INT(A, S), to mean that A has every intention of doing S and thus is committed to a course of action towards achieving S in the future.

Additionally, there are instances of actions:

1. PROC(A, M) refers to the action of A processing the KQML message M.
2. SENDMSG(A, B,M) refers to the action of A sending the KQML message M to B.

KNOW, WANT and INT approximately correspond to the psychological states of knowledge, desire and intention, respectively. Semantics for the *ask-if* performative are (Labrou and Finin, 1997):

ask-if(A, B, X)

1. A wants to know what B believes regarding the truth status of the content X.
2. WANT(A, KNOW(A, S)),
   where S may be any of BEL(B, X), or NOT(BEL(B, X))
3. **Pre(A)**: WANT(A, KNOW(A, S)) AND KNOW(A, INT(B, PROC(B, M)))
   where M is **ask-if(A, B, X)**
   Pre(B): INT(B, PROC(B, M))
4. **Post(A)**: INT(A, KNOW(A, S))
   Post(B): KNOW(B, WANT(A, KNOW(A, S)))
5. **Completion**: KNOW(A, S')
   where S' is either BEL(B, X), or NOT(BEL(B, X)).
6. **Pre(A)** and **Pre(B)** suggest that a proper advertisement is needed to establish them.

Pre(A) represents preconditions that have to hold for agent A (sender) to be able to send the message. **Pre(B)** are preconditions that have to be satisfied for agent B (receiver) to be able to successfully process the message. **Post(A)** and **Post(B)** are postconditions that hold for agents A and B, respectively, after successful utterance and processing of the message. **Completion** specifies the final state after possibly a whole conversation has taken place. Content of the illocutionary act of the message is specified under the point 2; for example, in the *tell* performative below, sender A says that it believes X to be true (BEL(A, X)).

Semantics for the *tell* performative are (Labrou and Finin,1997):

tell(A, B, X)

1. A states to B that A believes the content to be true.
2. BEL(A, X)

3. **Pre(A)**: BEL(A, X) AND KNOW(A, WANT(B, KNOW(B, S)))
   Pre(B): INT(B, KNOW(B, S))
   where S may be any of BEL(B, X), or NOT(BEL(B, X))
4. **Post(A)**: KNOW(A,KNOW(B, BEL(A, X)))
   Post(B): KNOW(B, BEL(A, X))
5. **Completion**: KNOW(B, BEL(A, X))
6. It is assumed that completion condition holds unless B sends error message, such is the case with any other performative.

We can see from the above cases of the message types that KQML semantics specify explicit preconditions and postconditions for both the speaker and the hearer. They show that it is assumed that the speaker has a model of the hearer's mental state; for example, the precondition for the sender **Pre(A)** in the above *tell* performative contains A's belief that B wants to know S. They also specify in a pragmatic manner what the hearer's mental state has to be in order to be able to interpret the message; hearer's recondition **Pre(B)** specifies that B has to intend to know S in order to understand A telling it about S. Those two points are crucial for the understanding of the message 'understanding', and they are therefore essential for the work of this thesis since our effort is to define the components of the interaction model needed to constrain, structure and interpret messages that agents exchange.

2.2.2 FIPA ACL Standardization effort

The Foundation for Intelligent Physical Agents (FIPA) is a non-profit organization that "was formed in 1996 to produce software standards for heterogeneous and interacting agents and agent-based systems" (FIPA, 2003).

Among other standards and recommendations, FIPA has the agent communication language standard named FIPA ACL. It provides complete specifications for the syntax and semantics. The semantics are stated as Feasibility Preconditions and Rational Effects. This approach acknowledges the fact that there is no guarantee that the expected outcome will actually hold after the message is sent, but the expectation of the effect taking place can guard the rational behavior.

FIPA ACL is important for the work in this thesis since we adopt it as the communication language for the agents that we developed and that we talk about, and, therefore, we will present it in more details. Some of the FIPA ACL performatives are *inform*, *confirm*, *disconfirm*, *not-*

understood, *query-if* and *query-ref*. In order to look into the structure of the FIPA ACL message, we present an example of a message:

```
(inform
        :sender         agent-i
        :receiver       agent-j
        :content
          "weather( tomorrow, snowing )"
        :language       Prolog)
```

Like KQML, a FIPA ACL message consists from a performative or a *communicative act*, and a set of parameters. Other common parameters not shown in the above example are 'ontology' and 'conversation-id'. 'Ontology' is, like 'language' parameter, referring to the content expressions. 'Conversation-id' is identifier of the ongoing exchange of the speech acts that form a conversation.

The descriptions of some of the FIPA ACL performatives are presented in the following text. The performatives are given with the following structure: *<sender, performative (receiver, proposition)>*.

<i, disconfirm (j, p)>

With the *disconfirm* message type, a sender is informing a receiver that the proposition *p* is false. The sender has to believe that *p* is false, and that the receiver either believes the proposition is true or is uncertain about it. The sender wants the receiver to believe that *p* is false.

<i, inform (j, p)>

With the *inform* message type, a sender is informing a receiver that the proposition *p* is true. The sender has to believe that *p* is true, and also that the receiver does not have any knowledge of *p*. This last condition might seem problematic since it looks like the sender needs to know about the receiver's knowledge of *p*, but it is not as strong requirement as explained in the following paragraph:

> The sender is *not* required to establish whether the receiver knows *p*. It is only the case that, in case that the sender already happens to know about the state of the receiver's belief, it should not adopt an intention to tell the receiver something it already knows. (FIPA, 2003)

<i, query-if (j, p)>

With the *query-if* message type, a sender is asking a receiver if a proposition *p* is true or not. The sender should have no knowledge whatsoever of the truthfulness of *p* and it believes that the receiver can answer if it knows the truth about the proposition. Here is the example of query-if message:

```
(query -if
        :sender agent-i
        :receiver agent-j
        :content
          "weather( tomorrow, raining )"
        :language Prolog)
```

The *query-ref* message type is a version of the querying speech act where the sender is asking from the receiver for the set of information objects that correspond to the given description.

<i, not-understood (j, a, p)>

By the *not-understood* message type a sender *i* informs receiver *j* that *j* just performed an action *a* which *j* did not understand. The most common case is for sender to *not-understand* the previously received speech act. The important issue for this thesis is how to understand the *not-understood* message and the reasons behind it. Here is the explanation:

> There may be several reasons for this: the agent may not have been designed to process a certain act or class of acts, or it may have been expecting a different message. For example, it may have been strictly following a pre-defined protocol, in which the possible message sequences are predetermined. The *not-understood* message indicates to the sender of the original, that is, misunderstood, action that nothing has been done as a result of the message. (FIPA, 2003)

The second part of the message content is proposition *p*. It represents the reason behind *not understanding* the action.

There is no guarantee that the reason is represented in a way that the receiving agent will understand. However, a co-operative agent will attempt to explain the misunderstanding constructively. (FIPA, 2003)

This section presented the FIPA ACL communication acts in the descriptive form. The formal semantics for the mentioned message types can be found in the Appendix A. The next section explores the topic of the conversation protocols and conversation policies.

2.2.3 From Messages to Protocols to Conversation Policies

Semantics of the isolated message types have limited reach in supporting the understanding of a message. In most cases, a message is a part of the larger context that consists from the other messages that preceded it, and the question is how to determine the meaning of a message within a conversation and the meaning of the conversation as the whole. For example, people find it obvious that an informative answer should follow the question, but the issue is how to encode rules like that in such a way that an agent can reason about it. In order to deal with this issue of sequencing the messages into a conversation, the ACL community moved towards conversation protocols. A conversation protocol is a set of the rules that define legal sequences of messages, and enables easier understanding of the meaning of the whole conversation.

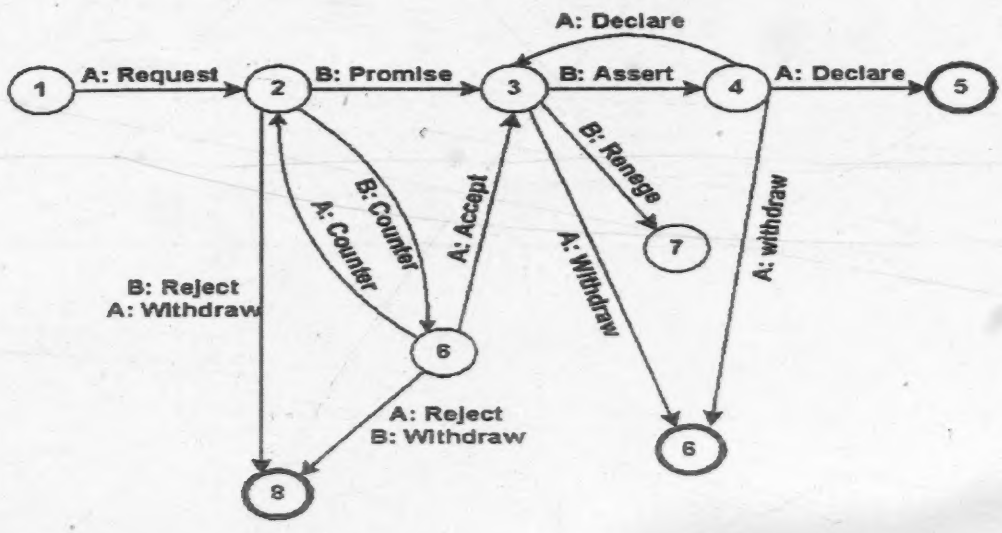


Figure 2.2 An example of the state transition conversation protocol (the basic conversation for action) (from Winograd and Flores (1986))

The fundamental notions underlying protocols can be traced to Winograd and Flores (1986). In an effort to analyze the meaning and the structure of the human language, they focused on the task-oriented conversations. Based on their observation in such conversations, they suggested that the crucial recurring pattern in a conversation is not based on the individual speech act but on the way speech acts related to each other. They called it a network of speech acts or a conversation.

According to this approach, the conversation protocol is modeled as the finite state automaton or the state transition diagram, as shown in Figure 2.2 (Winograd and Flores, 1986). This structure carries a resemblance to any software program, but the difference is that there is more than one actor engaged in making the transitions. Winograd and Flores (1986) also pointed in the direction of an additional important issue that needs to be investigated - how to link protocols with each other.

Various researchers have developed protocol level semantics such as the specifications of what mental states must hold before initiating the protocol, and what mental states should hold after completing the protocol (examples are Smith and Cohen (1996), Smith and Cohen (1998), Labrou and Finin (1997)). FIPA defines the so-called *interaction* protocols, such as Request Interaction Protocol Specification and (Dutch) Auction Interaction Protocol Specification (FIPA, 2003). But the conversation protocols do not resolve the whole problem of message and conversation understanding as elaborated in the next paragraph.

Researchers had high hopes for agent communication based on the following three components (Greaves et al., 2000):

1) Shared infrastructure for registration, reliable message delivery, etc. (structural interoperability)
2) Shared content ontology, truth theory, and method of binding objects to variables (logical interoperability)
3) Shared syntax and semantic of the Agent Communication Language (language interoperability)

It turned out that this was not enough for agents to be able to have meaningful conversations. Greaves et al. (2000) point to the fact that ACLs like FIPA ACL and KQML are logically powerful and expressive, and therefore it is not enough to agree on syntax of the performatives to be able to interpret the message. They call it the *Basic Problem*:

> Modern ACLs, especially those based on logic, are frequently powerful enough to encompass several different semantically coherent ways to achieve the same communicative goal, and inversely, also powerful enough to achieve several different communicative goals with the same ACL message. (Greaves et al., 2000)

Let us look at the simple example. If an agent receives a message *inform(database X crashed)*, it can reason about the sender's goal behind the message: sender may have needed the data from database X, but since it cannot get it from there, it might expect me to provide this data. Alternatively, the sender may believe that I need the data from database X and since I cannot get it, it believes that this is relevant for me to know this (concerning my task). Or perhaps it may be offering to give me this information, or this information can be relevant to an agent that I can talk to and the message sender cannot. The point is that there are possibly several intentions behind this *inform* message, and the receiver cannot know with certainty which one the sender had. Due to the *Basic Problem*, taken in combination with the complex conversational situations and the fact that the agents (usually) are resource bounded, "it is nearly impossible for an agent to reliably infer the intentions and goals underlying another agent's use of a particular ACL messages" (Greaves et al., 2000). Therefore, we need an additional mechanism to enable agents to successfully recognize the intentions and carry on with the conversations. Greaves et al. (2000) propose the notion of a conversation policy. The conversation protocols are considered as a crude subset of the needed conversation policy.

Additionally, the events and the state of the world in general also present the context for a message. More to the point, the world state is usually represented for an agent as its problem solving situation and the task semantics. Elio et al. (2000a) argue that conversation protocols simply move the issue of the meaning and understanding to a different level without resolving it. They take a different direction towards the notion of the cooperative speaker resolving the intention of a message in the context of the (shared) task knowledge, which is inspired by the computational linguistic approaches to the discourse understanding. These approaches will be discussed below in more details since they are essential for this work.

2.2.4 Summary

We started this section by enumerating some of the key similarities and differences between KQML and FIPA ACL. They have almost the same syntax, i.e. the messages from the

both languages look syntactically the same, except that the performatives have different names. It is important to note that this similarity is of the superficial nature. Another, more deep similarity is that semantics of the both languages assume the existence of some sort of the agent mental states and the existence of the models of the 'other' agents. But the differences between KQML and FIPA ACL semantics are significant "resulting in an impossibility of coming up with exact mappings or transformation between KQML performatives and their completely equivalent FIPA primitives, or vice versa" (Labrou et al., 1999). One of the key differences is the pragmatic approach of KQML towards semantics. KQML semantics specify the preconditions and the postconditions for the sender and for the hearer. Certain mental state (as expressed by preconditions) is necessary for a sender to send a message, and for a hearer to be able to accept and process the message. Postconditions specify the state of the sender after the message was sent, and the state of the hearer after receipt of the message (and before the answer). FIPA ACL semantics specify the *feasibility preconditions* as necessary beliefs or state of the sender, and the *rational effect* as a desired but not guaranteed perlocutionary effect of the communication action. Additionally, KQML also defines *completion* conditions that indicate the final state of the sender upon the completion of the conversation this particular message is part of.

The choice of performatives shows that FIPA ACL is seeking to be a 'pure' ACL, while KQML has more pragmatic approach. In this case, 'pure' denotes the ACL that relies more on the speech act theory principles, and KQML has performatives such as 'insert' and 'eos'. The message type 'insert' is closer to the database operation than to the act of speech. Similarly, 'eos' (end of stream) performative resembles the network protocol message. Both of these message types can have pragmatic value in some situations, but they are not based on the speech act theory.

The next step from the communication act is towards the conversation protocols that define the sequences of messages. But protocols do not completely resolve the issue of meaning and intention ambiguity. Greaves et al. (2000) suggest that additional knowledge is needed to define the meaning and constrain the possible intentions that a message may have. Elio and Haddadi (1999) argue that this additional knowledge, serving for the meaning and intention disambiguation, can be found in the context of the task the agents are jointly working on. It is important to note that the agents that are cooperating and communicating are talking *about something*. That something is their joint task, i.e. the progress they are making on the task, the information they need to proceed with their task, etc. The knowledge about the task structure and agents' roles within the task can be used to constrain the messages that can be generated and to enable the understanding of the messages that can be received. The next sections will review the

task directed dialogues and the discourse planning as part of the communication and cooperation issues.

2.3 Communication and Cooperation

This section discusses the cooperation and communication from different perspectives. First, we present the theoretical approaches towards the cooperation and coordination that recognize communication as the key element, namely joint intentions and joint responsibility concepts. Second, we discuss task based dialogue from the perspective of the discourse understanding community. This work is one of the key influential areas for this thesis, i.e. for grounding the agent interaction model in the task knowledge. Third, we discuss task modeling and communication through the review of the prominent work in this area, TAEMS and its connection to agent communication. The task modeling is another key concept used in this thesis. Fourth, we take a look at the multi-agent frameworks from the AI community perspective and we discuss their approach to cooperation and communication.

2.3.1 Joint Intention and Joint Responsibility

This section reviews theoretical approaches to agent cooperation and communication. *Joint intention* theory (Cohen and Levesque, 1991) models cooperation and coordination of agents working jointly on a task. It is based on the Cohen and Levesque (1990) work that provides formal constructs for individual intentions. There, the intention is described as agent's internal commitment to perform a certain action while having a particular mental state, i.e. it is defined through a goal that persists over time (this approach was described in more detail above). Joint intention theory starts with the notion of the team of agents jointly intending to perform an action, and it is concerned with the issues of what 'glues' these agents together into the joint commitment. Importantly for the work in this thesis and, more generally, for any work that is concerned with cooperating agents, Cohen and Levesque identify the communication as the focal point of the larger cooperation scenario:

> Communication required: Any theory of joint action should indicate when communication is necessary. What do agents need to know (and when) about the overall activity, about their own part, and about the other agents' shares? Should agents communicate when the joint action is to begin, when one agent's turn is over, when the joint action is finished, when the joint action is no longer needed? How

does communication facilitate the monitoring of joint intentions? (Cohen and Levesque, 1991)

Joint intention is the agent mental framework aiming to provide agents with the properties that would enable them to successfully work on a joint task. The notion of the joint intention is "formulated as a joint commitment to perform a collective action while in a certain shared mental state, as the glue that binds team members together" (Cohen and Levesque, 1991). Communication is introduced as the necessary part of the joint action and the agent (a team member) is committed to communicate with other team members if it acquires knowledge that makes the 'shared mental state' obsolete. Those cases are when a joint goal becomes impossible, achieved or irrelevant, and the agent is obliged to make this mutually known in the team for joint commitment is no longer needed. For example, if an agent solves a task and infers that a joint goal is thereby achieved, it will adopt an intention to inform the other involved agents about it, so that they can all believe that their goal is achieved and any lingering problem solving process can be dropped.

Jennings (1995) focused on the explicit model of cooperation and proposed *joint responsibility* concept as its incarnation. Joint responsibility concept is built on top of the joint intention notion. The key part that joint responsibility adds to joint intention theory is the idea of a common plan to achieve the joint goal. Jennings argues that explicit cooperation requires not only the common goal and the common intentions to achieve that goal, but also a *common course of action*, i.e., the common plan. In the implementation of joint responsibility (GRATE*), all involved agents have an actual representation of the joint intention that consists from the explicit model of joint actions. Additionally, each agent has more detailed representations of their individual intentions corresponding to the roles they are assigned in the joint intention schema.

In conclusion, the joint intention and the joint responsibility as theoretical approaches to the cooperation indicate two fundamental notions that are necessary for the successful cooperation, namely the joint commitment to communication and the common plan for joint action. Similar ideas have emerged in computational linguistics research that is concerned with dialogue understanding. The next section reviews this related research.

2.3.2 Task Based Dialogue

This section focuses on research concerned with models of dialogue understanding, particularly task based dialogues. These dialogues are about planning or performing a joint task.

The theories and models are primarily developed to account for human dialogue understanding, but many have been extended to human-agent dialogue modeling.

Generally speaking, the computational linguistic approach to dialogue understanding presumes that intentions (which drive utterances) are formed and understood within the context of the common knowledge about the joint task. As such, this approach is more similar to joint intention or joint responsibility theory, and notably unlike a predefined protocol approach to structuring a conversation.

2.3.2.1 Plan-based Speech Act Theory

Cohen and Perrault (1979) devised the plan-based theory of speech acts. They regard the speech acts as operators in a planning system where the problem-solving process consists of performing a sequence of actions that should result in achieving the goal state.

> Formal descriptions of plans typically treat actions as *operators*, which are defined in terms of applicability conditions, called *preconditions*, *effects* that will be obtained when the corresponding actions are executed, and *bodies* that describe the means by which the effects are achieved. (Cohen and Perrault, 1979)

Based on this, Cohen and Perrault illustrate of their theory by defining the operators for speech acts of informing and requesting. Their theory includes two main elements, namely the model of others and the model of plans. The model of others consists of beliefs and wants (goals) of other agents. They stress that a belief representation has to be able to include the belief that other agent knows the truth of a proposition p, and that at the same time the representation does not specify the actual truthfulness of p. This principle is shown by the AGT1's belief in line 3 in the example by Cohen and Perrault where agent AGT1 can have the following beliefs about agent AGT2:

1. AGT2 believes that the train leaves from gate 8.
2. AGT2 believes that the train has a departure gate.
3. AGT2 knows what the departure gate is for the train.

Thus, case 3 allows AGT1 to believe *that* AGT2 knows what the departure gate is without AGT1's actually knowing which gate AGT2 thinks that is. This

distinction will be useful for the planning of the questions. (Cohen and Perrault, 1979)

The models of plans are represented as sequences of operators. All operators have preconditions and effects. There are two types of preconditions. The 'CANDO' preconditions specify conditions that must be satisfied for the operator instance to be applicable. The 'WANT' preconditions reflect the principle of intentional behavior, i.e. agent has to want to do the action.

Cohen and Perrault build their speech act plans based on the Searle's theory. Their proposal for the INFORM and REQUEST plans is shown in Figure 2.3.

REQUEST(SPEAKER, HEARER, ACT)	
CANDO.PR:	HEARER CANDO ACT
WANT.PR:	SPEAKER BELIEVE SPEAKER WANT request-instance
EFFECT:	HEARER BELIEVE SPEAKER BELIEVE SPEAKER WANT ACT
INFORM(SPEAKER, HEARER, PROP)	
CANDO.PR:	PROP
WANT.PR:	SPEAKER BELIEVE SPEAKER WANT inform-instance
EFFECT:	HEARER BELIEVE SPEAKER BELIEVE PROP

Figure 2.3 REQUEST and INFORM plans

Cohen and Perrault suggest that "'Cando' preconditions and effects of speech acts should be defined in a way that does not depend on who the speaker of the speech act is. That is, no CANDO.PR or EFFECT should be stated as a proposition beginning with 'SPEAKER BELIEVE'" (Cohen and Perrault, 1979). Therefore CANDO and EFFECT in the REQUEST and INFORM plans contain only propositions starting with 'HEARER BELIEVE' or propositions with the neutral point of view (e.g. CANDO precondition in INFORM plan).

Intuitively speaking, the speech act is made with the intention to bring about a certain perlocutionary effect. Still, a speaker cannot ensure that the desired perlocutionary act will

actually happen. Accordingly, EFFECT in the above plans does not have the perlocutionary act itself, but it has the speaker's belief in the intended perlocutionary effect. This notion and related issues can be traced from here to the agent communication languages, namely KQML postconditions and FIPA ACL rational effect, as described earlier.

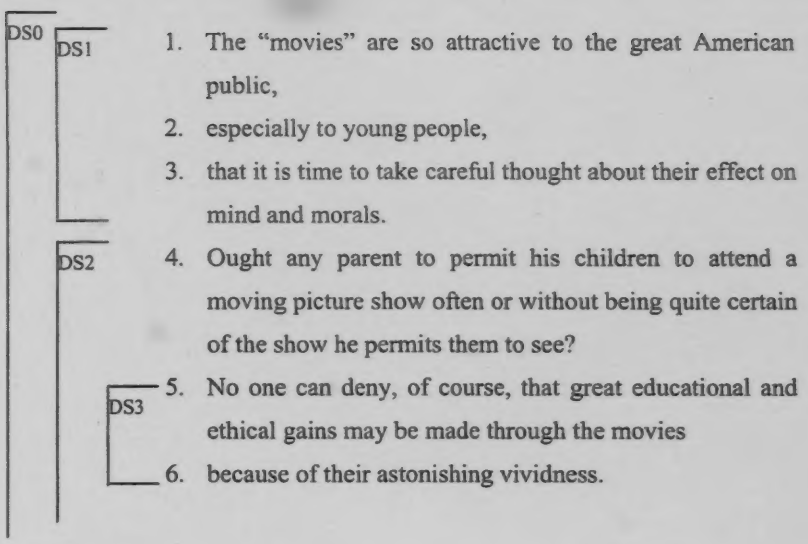
Cohen and Perrault made an interesting account of the speech act *side effects* that is important for this work. According to them "by the very fact that a speaker has attempted to perform a particular speech act, a hearer learns more – on identifying which speech act was performed, a hearer learns that the speaker believed the various preconditions in the plan that led to that speech act held" (Cohen and Perrault, 1979). The term side effect is referring to this belief acquisition by the hearer. The minimal side effects are the hearer's beliefs about speaker's belief that the speech act preconditions are satisfied. If we look at the INFORM plan in Figure 2.3, a hearer can conclude after receiving an *inform* message (with a proposition) that the speaker believes in the truthfulness of the proposition and that it wanted to generate the *inform* instance.

Cohen and Perrault's work is important because they presented the speech acts as something to be planned for in order to achieve the desired effect and, thus, they enabled the incorporation of speech acts as operators in planning systems.

2.3.2.2 Theoretical constructs for task related dialogues: SharedPlan

From the computational linguistics perspective, Grosz and Sidner (1986) have developed the theory of discourse or dialogue structure. It contains three components, each representing a discourse structure from the different perspective: the sequence of utterances, the structure of intentions, and the state of focus of attention. All of those three parts are interleaved and can be looked at as three dimensions of the discourse. The authors incorporated two non-linguistic components (the intentional structure and the attentional state) with the linguistic structure into a coherent theory of discourse.

The first element of the discourse theory is linguistic structure or the sequence of utterances. The linguistic structure consists of the sequence of utterances that are naturally grouped into discourse segments. A discourse segment (DS) can contain just utterances, it can have embedded other discourse segments, or it can have some combination of the both. To present the intuition behind this, here is the excerpt from the movie essay example from the Grosz and Sidner (1986). It contains the sequence of utterances grouped into the discourse segments (DS):


DS0
DS1
1. The "movies" are so attractive to the great American public,
2. especially to young people,
3. that it is time to take careful thought about their effect on mind and morals.
DS2
4. Ought any parent to permit his children to attend a moving picture show often or without being quite certain of the show he permits them to see?
DS3
5. No one can deny, of course, that great educational and ethical gains may be made through the movies
6. because of their astonishing vividness.

The second element is attentional state:

> Attentional state contains information about objects, properties, relations, and discourse intentions that are most salient at any given point. It is an abstraction of the focus of attention of the discourse participants; it serves to summarize information from previous utterances crucial for processing subsequent ones, thus obviating the need for keeping a complete history of the discourse. (Grosz and Sidner, 1986)

Attentional state is modeled as a stack that represents the dynamic focus during the discourse. It contains focus spaces, and each focus space (FS) is associated with a discourse segment.

The third element is intentional structure. It is this structure of intentions that makes a discourse coherent. Intuitively speaking, the discourse purpose is the top-level motivating intention that initiated the discourse in the first place. For example, a discourse can have a purpose to 'come to know when the train leaves from Calgary'.

The intentional structure is closely related to the linguistic structure. Each discourse segment (DS) has a discourse segment purpose (DSP). We can observe a discourse from intentional and linguistic perspective, and we can see that as the discourse is built from discourse segments in the equivalent manner, there is also (top) discourse purpose with sub-purposes. A

discourse segment purpose that contributes to the top purpose is dominated by that top purpose and this hierarchy is called *dominance hierarchy*. For example, a top discourse purpose to 'come to know when the train leaves from Calgary' can have a sub-purpose to 'convince another person/agent to intend to contact the train station'. An example of the linguistic structure, and hierarchy of segments and segment purposes is depicted in Figure 2.4 (left and right part) (adapted from Grosz and Sidner (1986)), the middle part shows the focus space (FS) stack connected with the corresponding discourse segments.

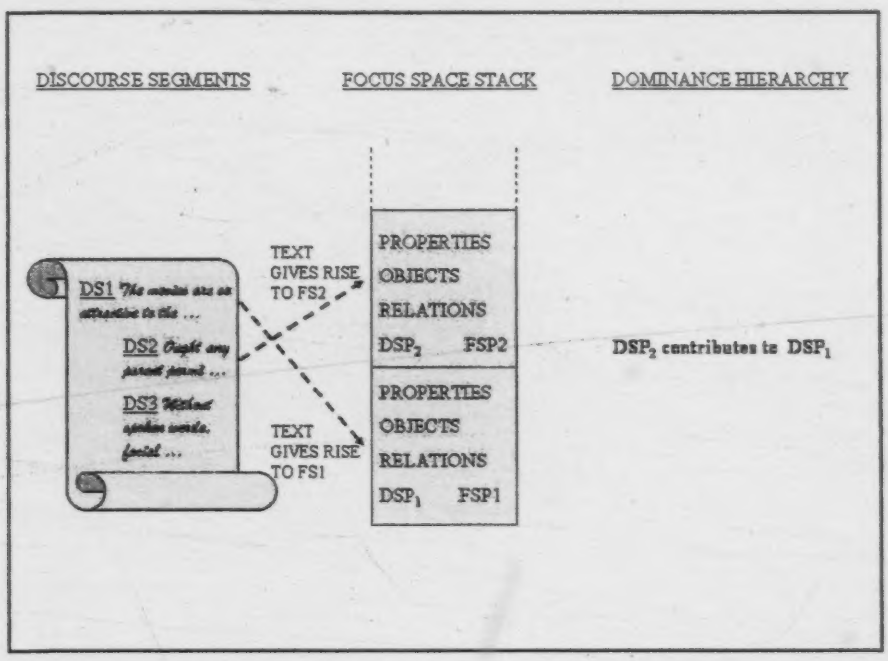


Figure 2.4 A discourse structure illustration

The Figure 2.4 shows how a linguistic structure along with the equivalent intention structure influences the state of the focus space stack (adapted from Grosz and Sidner (1986)). The structure of intentions in the Grosz and Sidner discourse structure theory is the basis for the formal model of collaborative plans named SharedPlan (Grosz and Kraus, 1996). It points in the direction of the relation between a shared task model and the intentional structure of the discourse that spawns from it, which is crucial to this thesis.

SharedPlan (the original version presented in Grosz and Sidner (1990); new improved version in Grosz and Kraus (1996)) is introduced with the intention to model collaborative behavior and task-oriented dialogues. The structure behind it is deduced from the analysis of

human collaborative behavior and communication. If two persons work on the task cooperatively, they must have a shared plan about how to do it.

High level representation of the SharedPlan model is the following (Grosz and Kraus, 1996):

To have a collaborative plan for an action, a group of agents must have:

1. mutual belief of a (partial) recipe [plan]

2.a individual intentions that the action be done

2.b individual intentions that collaborators succeed in doing the (identified) constituent subactions

3. individual or collaborative plans for subactions

Firstly, SharedPlan must have the actual 'shared plan', i.e. the plan (recipe) known to all involved agents. Secondly, agents do not only have to commit to their individual intentions, but also to group intentions and individual intentions of their collaborative partners. Such reasoning is responsible for the helpful behavior in the context of collaborative process. And thirdly, group activity plans can consist from individual plans or other collaborative plans in a recursive manner.

A SharedPlan is the collaborative equivalent of the individual plan; it actually includes individual plans as its components. SharedPlan also requires a group to have mutual belief that all members are committed to the success of the group action, mutual belief that subactions need to be performed either by a group member or a subgroup.

When an agent has a partial plan (shared or individual) it uses communication with collaborative partners to fill in the missing information. Therefore, SharedPlan not only formalizes a shared knowledge concept in the form of mutual beliefs about goals and actions, capabilities, intentions and commitments of others, it also prescribes problem solving driven communication. In other words, the communication structuring and understanding is strongly grounded in the joint problem solving.

On the conceptual level, SharedPlan is a formal model that is trying to capture all the complexities of the collaborative behavior and related communication; the other approaches to these issues, joint intention and joint responsibility theory, have notable similarities in stressing the importance of the common plan and communication.

2.3.2.3 Plan based models of dialogue

This section looks into the different types of plans, such as domain specific plans and discourse plans, and how they contribute to the plan based model of dialogue.

Litman and Allen (1990) proposed a plan based approach to discourse; they made the distinctions between some important concepts for the plan based dialogue model. Firstly, they distinguished the discourse intentions from the commonsense plans. The intentions are used in the discourse by a speaker to refer to the commonsense plans. For example, the discourse intention of a person talking to a bus station clerk may be to make the clerk believe that s/he wants to buy a bus ticket to Calgary. A commonsense plan can describe how to take a trip by bus (which includes buying a ticket). The above mentioned discourse intention refers to the part of the commonsense plan responsible for buying a ticket. This example shows that although the discourse intention and the commonsense plan are connected, they represent different concepts. Secondly, the commonsense plans are divided into the domain plans and the discourse plans, where the domain plans are used for task modeling, and the discourse plans denote domain-independent plans. For example, the discourse plans are responsible for generation of dialogue interruptions such as clarification and correction subdialogues.

The tripartite plan-based model of dialogue by Lambert and Carberry (1991) enhances the model proposed by Litman and Allen (1990) by further distinguishing between plan categories independent of the task domain. Litman and Allen (1990) have one type of non-domain plans they called the commonsense discourse plans but Lambert and Carberry (1991) argue that it really conflates two very different types of plans into one. Those two different plans are the following: the first one is the general problem solving type of a plan that is used for handling the domain plans but it is not dependent on the domain itself; the second one is the category of discourse plans that are used exclusively for communicative actions.

The goal of the problem solving plans is a metagoal, according to Lambert and Carberry (1991). Its purpose is manipulation (e.g. construction and reparation) of domain plans. Here is an example:

> For example, if an agent has a goal of earning an undergraduate degree, the agent might have the problem-solving goal of selecting the instantiation of the degree parameter as BA or BS and the problem-solving goal of building a sub-plan for satisfying the requirements for that degree. (Lambert and Carberry, 1991)

The problem solving plans also play a role of a mediator between the domain and the discourse plans. If a domain plan is being constructed by a problem solving plan and information is needed for the domain plan to be complete, the problem solving plan will invoke a discourse plan to obtain the information, and it will eventually provide the domain plan with the received information. Intuitively speaking, "a *discourse goal* is the communicative goal that a speaker has in making an utterance, such as obtaining information or expressing surprise" (Lambert and Carberry, 1991).

2.3.2.4 The Knowledge Preconditions

In order to be able to perform a given action, an agent needs to satisfy preconditions for that action, whether they are related to a physical or a knowledge state. In Lochbaum (1995), the concept of the knowledge preconditions is introduced into the SharedPlan framework for discourse.

The SharedPlan model uses operators related to the group and individual beliefs about abilities to perform actions. The purpose of these operators is to specify the conditions under which an agent is capable of performing an action. Still, their definition requires satisfaction of only the physical preconditions or the action performance constraints. Lochbaum (1995) takes steps to fix the situation with the knowledge preconditions:

> Because an agent is not truly capable of performing an act unless it possesses the appropriate knowledge, the [SharedPlan operators'] definitions must be augmented with the axiomatization of knowledge preconditions.

Lochbaum introduces two axioms. The first axiom refers to the recipes and formalizes the following:

- An agent needs to know a recipe (plan) for the action it has to perform.
- An agent does not need recipes for the basic-level actions, they are executable at will.

The second axiom refers to the parameters and states:

- An agent must be able to identify the parameters needed by an action it is about to perform.

If any of these knowledge preconditions are not satisfied, an information seeking dialogue can be generated, and it should provide the agent with the needed knowledge prerequisites. As an example, Lochbaum gives an episode of the dialogue between persons involved in removing a flywheel. A person who should remove the flywheel at first does not know the way to do this, and then s/he needs to loosen the screws but does not know where to find one of them. Since the axioms from above are not satisfied, the information seeking dialogues are started (from Lochbaum (1995)):

(1) E: First you have to remove the flywheel.
(2) A: How do I remove the flywheel?
(3) E: First, loosen the two allen head setscrews holding it to the shaft, then pull it off.
(4) A: OK.
(5) A: I can only find one screw. Where's the other one?
(6) E: On the hub of the flywheel.

The author shows how the requirements for the knowledge preconditions can be used to explain the information-seeking subdialogues. This is done in the environment consisting from the union of SharedPlan model (Grosz and Kraus, 1996) and the theory of discourse structure (Grosz and Sidner, 1986). Lochbaum draws the explicit connection between these two concepts:

> Each segment of a discourse has an associated SharedPlan. The purpose of the segment is taken to be intention that ... the discourse participants form that plan. This intention is held by the agent who initiates the segment. (Lochbaum, 1995)

Based on that, utterances of the discourse can be understood through recognizing the contribution of the utterances towards a SharedPlan paired with the discourse segment. This mechanism resembles the conversation protocols since utterances have to follow certain rules in order to move the SharedPlan from one state to another.

2.3.2.5 Collagen

Collagen (Rich and Sidner, 1997) is a collaborating agent toolkit. Rich and Sidner (1998) claim that it is application-independent collaboration manager that stands between the agent and

the outside world. The outside world is primarily represented by humans since overall goal of the research done on Collagen is to enhance "human-computer interaction which explicitly supports the user's problem-solving process" (Rich and Sidner, 1998). Collagen's collaborative behavior is based on theories of collaborative discourse: theory of discourse structure by Grosz and Sidner (1986) and SharedPlans (Grosz and Sidner, 1986; Grosz and Kraus, 1996).

Collagen (Rich and Sidner, 1998) uses a Plan tree as the implementation representation of the SharedPlan. The Plan tree consists from "alternating act and recipe nodes" (Rich and Sidner, 1998), where recipe denotes a plan. Each act node has its parameters, a performer, and a recipe if it is a non-primitive; and a recipe node has a binding to the acts (act nodes) that correspond to the steps in the recipe.

As described in Rich and Sidner (1998), Collagen's internal discourse state representation consists from a History list and a Focus Stack in addition to the Plan tree. The focus stack holds current segment of the discourse, and the History List shows top-level discourse segments popped off the Focus stack. An example of the internal discourse state representation is shown in Figure 2.5.

The focus stack is an implementation of the attentional state that is a component of the theory of discourse structure developed by Grosz and Sidner (1986). Attention focus dynamics of the communication flow are represented by pushing and popping discourse segments to and from the Focus stack, where "a segment is a contiguous sequence of communication acts that serve some purpose" (Rich and Sidner, 1997). Rich and Sidner (1997) have made the connection between the SharedPlan and the focus of attention notion by assigning to each discourse segment a SharedPlan for its purpose.

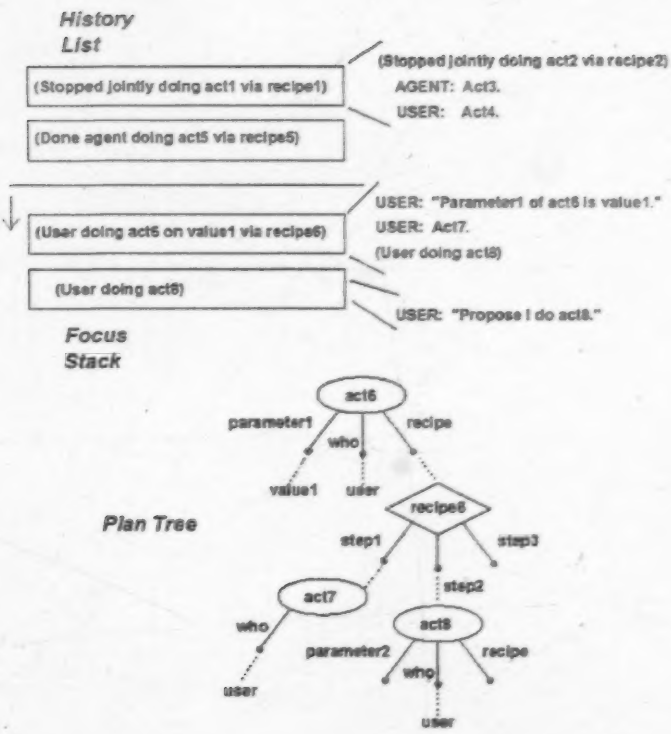


Figure 2.5 Collagen: Internal discourse state representation (from Rich and Sidner (1998))

The SharedPlan concept is crucial in the Collagen's discourse management that handles discourse generation and understanding. It is used to determine what are the plausible speech acts from the human user at a given moment, and then to constrain the interface so that only those are available. It is also used to resolve the incoming speech acts in terms of the active plans.

Rich and Sidner (1997) describe their "discourse generation [as an] algorithm [that] looks at the current focus stack and associated SharedPlan and produces a prioritized agenda of (possibly partially specified) actions which would contribute to the current discourse segment purpose" (Rich and Sidner 1997); those actions can be communicative actions.

The agents' speech acts, as shown in Rich and Sidner (1998), are primarily generated because of two essentially different reasons: a) generic discourse principles and b) application-specific knowledge.

Generic discourse principles are based on Lochbaum (1995) knowledge preconditions: 1) knowledge about a plan for the task is needed, 2) other participant can be asked to identify a plan,

3) in order to perform an action all needed parameters have to be known, and predecessor actions (in the current plan) have to be performed.

Some of the agent's utterances are coming from the application-specific knowledge. The application-specific algorithm might be triggered by a certain situation to produce a speech act. An example by Rich and Sidner (1998) shows a case where a user is trying to come up with an air travel route but s/he gave too many constraints which produced zero routes. The application-specific algorithm recognized a situation and offered to relax a certain constraint that would give a user the applicable routes.

Collagen's discourse interpretation is essentially the inverse of the generation. The Collagen framework unites theoretical constructs such as the SharedPlan and the knowledge precondition[illegible]ystem that solidly grounds the communication into the shared problem solving context. But, it simplifies the communication act understanding and it avoids deviant communication acts by constraining the acceptable speech acts through the user interface, i.e. a user (input) communication is limited to the selection of one of the items (sentences) from the user menu (as shown in the lower left corner of the Figure 2.6).

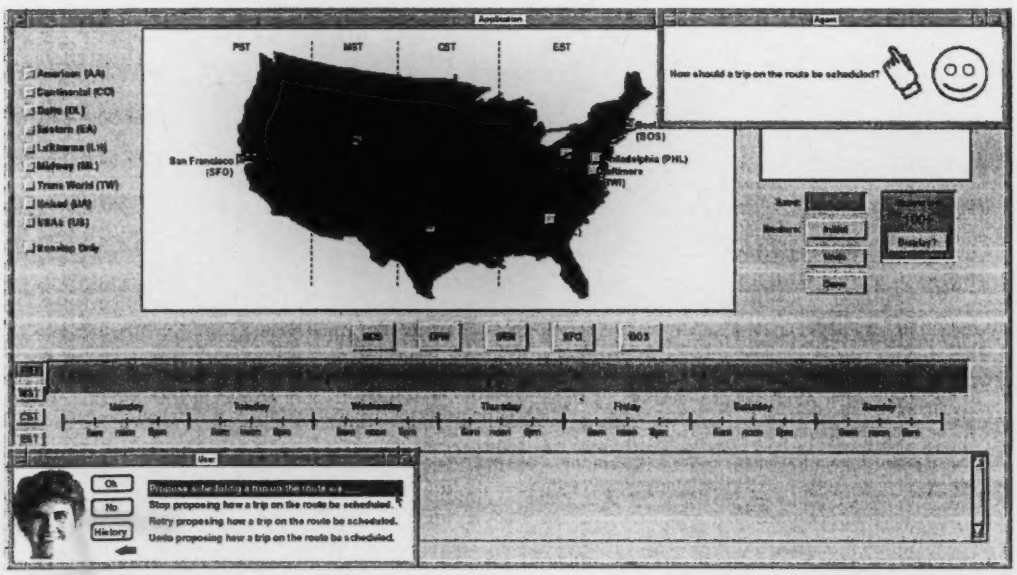

PST
MST
CST
EST
How should a trip on the route be scheduled?
Ok
No
History
Stop proposing how a trip on the route be scheduled.
Retry proposing how a trip on the route be scheduled.
Undo proposing how a trip on the route be scheduled.

Figure 2.6 Collagen test application screen (from Rich and Sidner (1998))

2.3.2.6 The Abstract Task Model

The Abstract Task Model (Elio and Haddadi, 1999; Elio and Haddadi, 2000; Elio et al., 2000a; Elio et al., 2000b) is an effort to deal with the basic problem of the communication (Greaves et al., 2000) of many-to-many mappings between agent's intentions and message primitives. The conversation protocols based on finite state transitions are not the complete solution for conversations:

> ... there are two drawbacks to the adoption of a state-transition approach to conversation structure. First, we believe it is insufficiently rich for specification of the messages that can be exchanged and the conversation policies that govern them. Secondly, for dialogue-intensive domains it is impractical to know all the possible paths of a conversation in advance and represent them as a protocol. (Elio and Haddadi, 1999)

This research was motivated by the application that includes human-agent dialogue and therefore discourages the assumption that a human user will necessarily know the implemented conversation protocols or even be willing to follow it.

Abstract Task Specifications assume the usage of the conversation protocols only to maintain *local coherence*, i.e. unity between sequences of 2-3 messages. Apart from the messages that conform to the protocols, there must be a way to infer the intention behind the unexpected messages due to the reasons mentioned above. Elio et al. (2000a) "argue that the way to constrain messages is to constrain intentions". They propose Abstract Task Model (ATM) to constrain agent's task and discourse intentions. ATM gives agent a meta-level ontology by defining:

> (i) what the agents can talk about (the objects of discourse),
> (ii) what they can say about those things, and most importantly,
> (iii) how to progress on the task and when the task is completed.

This ontology enables agents to talk about tasks independent of the particular task domain. It also enables run-time recognition of the intentions in a conversation related to a joint task.

ATM promotes the idea of task and discourse being represented as two separate transition spaces. When no progress can be made in the task space, a discourse intention is spawned and

communication actions (operators) are performed in the discourse space. For example, if my task intention requires the knowledge of *p* (which I do not have), I may set a discourse intention to query somebody else about *p*, and on completion I would return to the task space. Analogously, when no progress can be made in the discourse space, a task intention is generated and task operators are executed. For example, if I receive a query about *p* and I do not know *p*, I may set a task intention to somehow compute *p* and, afterwards, to return to the discourse space. The conversation protocols are viewed as operators in discourse space. In the task space, task protocols are used to advance through state transitions, they serve as operators.

Overall, ATM supports the ability to deal with the messages within the protocols or the messages that breach protocols but conform to the general intention structure specified by the Abstract Task specifications. There is an open issue what to do with a message that does not belong to either of these cases, i.e. when a message cannot be understood within the joint intentional structure defined by the Abstract Task Model. In this thesis, we advocate the view that some sort of a model can be used to define those "not-understood" cases, i.e. the communication "error conditions".

2.3.2.7 Summary

In this section, we discussed the notion of the speech acts being perceived as the operators and as something that can be planned. Various approaches and systems (SharedPlan, Plan-based discourse, Collagen, ATM) propose an intimate relation between the discourse structure and the discourse intentions, and the task intentions. The key idea in this approach is that the speech act intentions can be resolved in the context of the task structure and the agent roles. Overall, the motivating notion for this thesis is the proposal that the communication actions are operators that advance task progress. The next section looks more closely into the task modeling and its relation to the communication.

2.3.3 Task Modeling and Communication

Task modeling, and how it relates to the communication, has a central position for this work. Since the review of the whole field is beyond the scope of this thesis, we review very influential work in this area, namely TEAMS, and investigate its approach to agent communication.

2.3.3.1 TAEMS framework

TAEMS (Task Analysis, Environment Modeling, and Simulation) is the framework for simulating and analyzing behavior of single agents or multi-agent systems while they are performing problem solving actions (Decker and Lesser, 1993; Decker et al., 1996; Decker, 1996; Lesser, 1998; Wagner et al., 1999; Lesser et al., 1999).

TAEMS is not an agent-centric framework, it is primarily focused on the task environment. It has an explicit and elaborate task representation that is independent of any agent architecture or the particular task domain. Decker and Lesser (1993) claim that TAEMS "formally captures the range of features, processes, and especially interrelationships that occur in computationally intensive task environments" to the extent that no previous research work did. TAEMS framework proved to be quite a powerful tool for simulating the multi-agent systems working on the tasks with intricate subtasks' interdependencies.

TAEMS project addresses the coordination issues in multi-agent systems. What kinds of problems are to be solved using TAEMS framework depends on what kind of environments can be represented and simulated using TAEMS. According to Decker and Lesser (1993), task environment in TAEMS refers to "a problem domain in which the primary resources to be scheduled or planned for are computational processing resources of an agent or agents, as opposed to physical resources such as materials, machines, or men". The authors give the examples of such environments: distributed sensor network (where sensors are not the primary scheduling consideration), distributed design problems, complex distributed simulations, control processes for almost any distributed or parallel AI application, and intelligent environment control applications.

Considering how TAEMS framework is task oriented, it is not surprising that it has such a detailed and rich task structure. TAEMS task environment represents task decomposition on three different levels: the generative level, the objective level and the subjective level. The generative level is an important part for the task environment simulation, since it is responsible for the generation of the two other levels. The objective level describes the "essential, 'real' task structure of a particular problem-solving situation" (Decker & Lesser, 1993), i.e. it is a complete description of a given task and its environment. The subjective level is dependent on a particular agent and it determines what an agent can see from the whole Objective level, i.e. it is the agent's local view. Additionally, an agent can have the subjective interpretation of values from the objective task structure.

Let us look at what is TAEMS task model exactly:

> TAEMS task models are hierarchical abstractions of problem-solving processes that describe alternative ways of accomplishing a desired goal; they represent major tasks and major decision points, interactions between tasks, and resource constraints but they do not describe the intimate details of each primitive action. (Wagner et al., 1999)

In order to investigate closer what a TAEMS task structure consists of, let us look at the Objective level. Although TAEMS framework can deal with multiple task groups (or problem solving situations) we will look here at a single instance of a problem solving situation. The objective level has the problem solving situation decomposed into a directed acyclic graph. The graph consists of the root task that is connected to its subtasks, we will also refer to a task as T. Those tasks can have their own children nodes, called subtasks. A task without subtasks is an executable method, also called method M. This is the skeleton of the TAEMS task representation on the Objective level. The real meat of this system are the different kinds of relationships between the nodes in the graph, plus the functions and the values associated with the nodes of the graph.

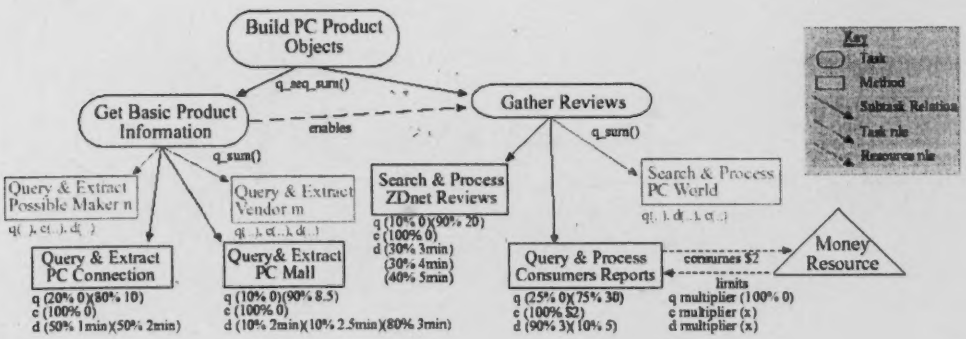


Figure 2.7 A representation of a simple TAEMS task model (from Wagner et al. (1999))

There is a formal 'subtask' relationship defined between a task and all its direct subtasks. This relationship is connected to a *quality function* Q (or quality-accumulation-function *qaf* or *q*). For a task T, quality function Q is determined by the quality function values of its direct subtasks. What exactly is the connection depends on how the problem solving situation is modeled, quality

can be determined for example using *minimum*, *maximum*, *average* or *sum* of subtasks' quality values. A quality function also determines how is a task T decomposed. The maximum quality function is equivalent to the OR decomposition, and minimum quality function is equivalent to the AND decomposition. There are also quality functions that specify task orderings. An example of a TAEMS task model is presented in the Figure 2.7 (Wagner et al., 1999). It shows the simplified sub-graph of the task structure used to construct product models of retail PC systems. The example of the quality function for the "Build PC Product Objects" task is *q_seq_sum* that specifies that both subtasks have to be performed, in order, and that their resulting qualities have to be summed to produce the quality of the parent task. Apart from the subtask relations, there are other task relations such as *enables* which will be described later as *non-local effects*. Each of the methods in the graph is described in terms of its expected quality (q), cost (c) and duration time (d).

In order to calculate the quality for all nodes in the task graph, recursive algorithm drills down towards the methods to perform its calculations. The quality function for a method is usually modeled as a linear growth function that grows as the method execution progresses and the maximum it can reach is the *maximum quality*, which will be elaborated a bit later. A whole task group, tasks and methods can have specified hard deadlines, after the hard deadline quality function is not increasing any more.

If we look from the perspective of agents that are situated within TAEMS task environment, we can see that the crucial concept is the Subjective level since it is the description of what an agent sees from the Objective level and what options are available to an agent in its interactions with the environment. For example, an agent could believe that a maximum quality of a certain method M is twice its objective value.

Apart from the obvious 'subtask' relationship, there are other relationships among tasks and methods. A task can have a *non-local effect* (nle) on another task/method, which means that it can influence the execution and performance of that other task. In other words, the execution of one task can affect another task/method's maximum quality or duration. Each method has defined an *initial maximum quality* value and an *initial duration* value, i.e. without any interference or interruption a method would finish executing after a time interval of length 'initial duration' and its quality function would reach value 'initial maximum quality'. But the non-local effects, i.e. the execution and the results from some other method or task (beside a direct parent task or subtask) can influence the method's maximum quality and duration in a positive or a negative manner, where the positive manner means increasing the maximum quality or decreasing duration, and the negative manner is described vice versa.

TAEMS framework has the possibility to support the models of complex task environments by providing a series of different classes of the non-local effects. Some of the commonly used non-local effect classes are presented in the following text:

Enables: If task T 'enables' method M, it means that M's maximum quality will be zero until the result from T is available. Then the M's maximum quality changes to initial maximum quality.

Facilitates: If task T 'facilitates' method M, and the result of T is available to M before the start of its execution, then the M's duration will be decreased or its maximum quality will be increased.

Hinders: This effect increases the duration and decreases the maximum quality of the hindered method M.

Precedence: 'Precedence' effect is similar to the 'enable' effect. If task T 'precedes' method M, the M's maximum quality will be zero until some quality is accrued for T. How the M's maximum quality is changing depends on the task's quality function and its maximum quality.

In case there are some non-local effects between the methods executed by different agents, they are called *coordination relationships*. For example, if tasks "Get Basic Product Information" and "Gather Reviews" (Figure 2.7) are to be executed by different agent, then the 'enables' non-local effect between them will also be the coordination relationship. In this case the non-local effects are dependent on the agents communicating the status and the results of a method or a task causing the non-local effect.

This kind of a task representation that is focused on the duration and quality of subtasks gives the possibility to make trade-offs between time and quality of the results for the problem solving situation. But there is still a question what is the role of agents in this task-centric environment.

In order to see what kinds of roles the agents play in the TAEMS framework we should start with exploring the expectations TAEMS has from the agents. As we have seen in the above text, TEAMS task models are mostly independent of any agent architecture. There are also not many assumptions on the agents' mental state. What is required from an agent is to have beliefs about the task environment, and to be able to perform the actions that enhance problem solving in the context of the given task environment. On the surface it looks like the basic requirements are low enough so that a process can qualify under the TAEMS definition of an 'agent'. But on the closer inspection it shows that this definition is actually dependent on the complexity of a task environment. If the task environment is complex enough that it requires complex coordination

mechanisms and a significant amount of high-level inter-agent communication, then the concept of 'agent' suddenly requires abilities for the complex rational and social behavior, and for the discourse generation and understanding.

A component that supports the agent's ability to interact with other agents and coordinate joint activities is General Partial Global Planning (GPGP or the more recent version GPGP2). Wagner et al. (1999) give the following definitions that connect the notions of conversation and coordination:

> ... we will use the term *coordination protocol* to describe the specification for a dialogue between one or more agents that is held for the purpose of coordinating their activities; a *conversation* is an instantiation of a protocol. A *coordination mechanism*, in contrast, denotes a larger grouping of concerns – it is the way in which an agent reasons about interactions, plans to resolve them, and carries out communication activities to do so. (Wagner et al., 1999)

The TAEMS conversation protocols are specified by means of the finite state automaton model. An example of the conversation protocol is shown on Figure 2.8. It consists from the conversational states and the transitions that are associated with calls to communication actions or analysis code. The conversation protocols are used to communicate the results, and to handle the task interrelationships. For example, through a conversation an agent ag1 will try to establish the commitment of another agent ag2 to perform the task that enables an ag1's task. Therefore, communication is used to enable the problem solving advancement. TAEMS does not use an ACL for the agent communication since it is a closed system of homogenous agents.

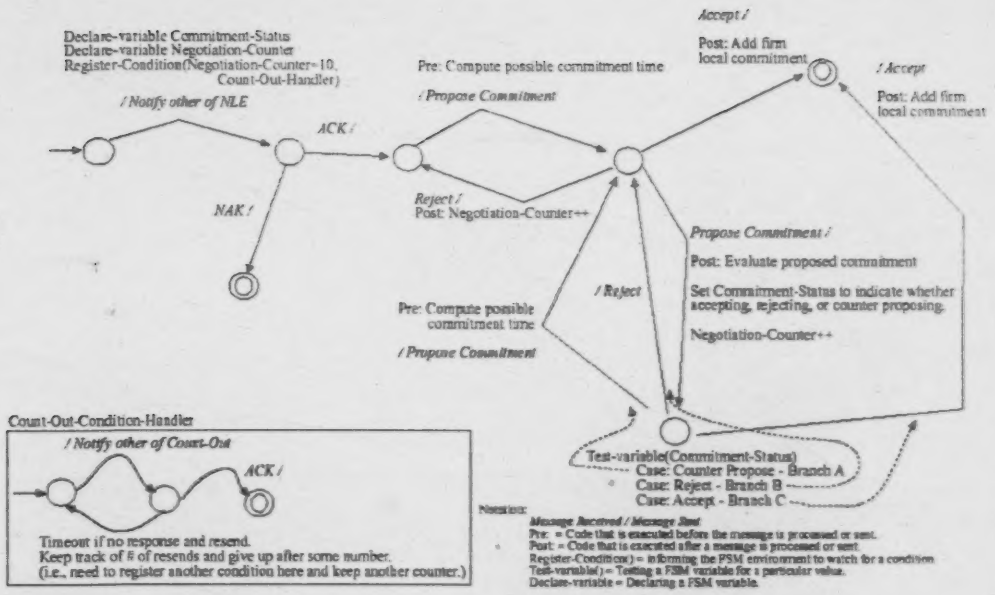

Declare-variable Commitment-Status
Declare-variable Negotiation-Counter
Register-Condition(Negotiation-Counter=10,
Count-Out-Handler)
/ Notify other of NLE
ACK /
NAK /
Pre: Compute possible commitment time
/ Propose Commitment
Reject /
Post: Negotiation-Counter++
Accept /
Post: Add firm
local commitment
/ Accept
Post: Add firm
local commitment
Propose Commitment /
Post: Evaluate proposed commitment
Set Commitment-Status to indicate whether
accepting, rejecting, or counter proposing.
Negotiation-Counter++
/ Reject
Pre: Compute possible
commitment time
/ Propose Commitment
Test-variable(Commitment-Status)
Case: Counter Propose - Branch A
Case: Reject - Branch B
Case: Accept - Branch C
Count-Out-Condition-Handler
/ Notify other of Count-Out
ACK /
Timeout if no response and resend.
Keep track of # of resends and give up after some number.
(i.e., need to register another condition here and keep another counter.)
Notation:
Message Received / Message Sent
Pre: = Code that is executed before the message is processed or sent.
Post: = Code that is executed after a message is processed or sent.
Register-Condition( ) = informing the FSM environment to watch for a condition.
Test-variable( ) = Testing a FSM variable for a particular value.
Declare-variable = Declaring a FSM variable.

Figure 2.8 A TAEMS conversation protocol example (from Wagner et al. (1999))

The TAEMS approach enables simulations and analysis of different sorts of situations to determine the problem solving and social behavior of agents in the dynamic and partially accessible but controlled environments. TAEMS has very detailed task structure that is rich and explicit in the representation of the task interdependencies. TAEMS agents' communication abilities are based on the coordination protocols defined through the finite state automaton transitions, which support a limited complexity of conversations and do not specify how to understand or how not to understand the meaning of messages.

2.3.4 Multi-agent Frameworks

The previous two sections have dealt with the task based dialogue and the task modeling, both of those approaches assume that agents have some sort of the *deep knowledge* about each other and each other's joint tasks. Most of the work we reviewed so far has focused on the intentions that underlie the planning and performing a speech act, and what knowledge the agents must or should have about each other.

A large part of multi-agent systems has adopted a protocol-based approach to the communication. This approach is coupled with the *shallow agent models* and (relatively) simple

cooperative tasks. We briefly review two multi-agent frameworks InfoSleuth and Retsina in order to contrast this approach to the previously reviewed research.

2.3.4.1 InfoSleuth and Retsina

InfoSleuth is an agent architecture for heterogeneous information retrieval, monitoring and analysis in dynamic environments (Nodine and Unruh, 1998; Nodine and Unruh, 2000). InfoSleuth agents do not posses any significant knowledge about other agents's state or knowledge. In order to enable such agents to communicate, the needed support elements are defined, namely the common set of speech acts, the common service ontology for describing agents' capabilities and relating them to a specific task execution, and the shared set of the prescriptive conversation policies that determine allowed message exchanges between agents. As a communication language, InfoSleuth framework uses a version of the KQML. Its conversation policies are embodied in the form of the conversation protocols "represented as finite-state automata, where the initial performative determines the conversation policy to use" (Nodine and Unruh, 2000).

One of the problems with the protocol based communication is that if the protocol wants to account for all the messages that can be received, including the improper messages, it will grow into very complex finite state transition diagrams. In order to avoid this, InfoSleuth has a component oriented approach to the conversation protocols. A particular conversation protocol should handle only the proper exchange of messages. In order to handle errors and other deviations in a conversation there are defined ways to extend basic protocols using extension components. Additionally, the conversation protocols can be concatenated into larger conversations. Overall, Nodine and Unruh (2000) are trying to resolve some of the problems of the protocols (as noted by Greaves et al. (2000)). This approach works with the agent's shallow knowledge of other agents, but it is limited to the simple cooperative tasks and simplistic level of communication such as elementary information exchange.

Retsina is an open multi-agent system infrastructure (Sycara, Paolucci and Shehory, 2000; Sycara et al., 2000; Sycara et al., 2001). Since it is an open system, different kinds of agents can join (and leave) the multi-agent framework. In such circumstances it is hard to assume that the agents would have models of each other or significant knowledge of each other's tasks. Therefore, Retsina also utilizes the shallow knowledge of other agents. Retsina agents need the common set of speech acts (KQML in this case), the common ontology and the common conversation protocols. Additionally, Retsina strives to enable agents to participate in the

complex interactions, and therefore it provides the infrastructure of services such as finding the other agents and mapping between the capabilities and the agents. In our opinion, this approach does not resolve the problems accompanying the complex agent interactions, it pushes the problem towards the services of the multi-agent system infrastructure and it assumes that the agents will have the 'hard-coded' knowledge about those services and how to use them.

2.3.4.2 Comparison between the (AI) Multi-agent Systems and the Task Based Dialogue Frameworks

So far we have looked into the multi-agent and the human agent/computer agent interaction frameworks with different approaches to the issues of agent cooperation, coordination and communication. There is an agreement that the communication is the key issue within the larger cooperation and coordination picture. All research efforts point to the fact that the set of shared infrastructure, common agent communication language, and common content language and ontology is not enough to support the communication needed for the successful cooperation and coordination. There are two main issues coming out of this situation. Firstly, how much knowledge do the agents really need to engage in cooperative dialogues? How should this knowledge be structured and formalized? In order to deal with this problem, we have to main directions:

- the open multi-agent systems, where agents have shallow models of each other (e.g.: InfoSleuth, Retsina)
- the multi-agent frameworks, where agents have significant shared knowledge about the problem solving situation and each other's roles and capabilities (e.g.: TAEMS, theoretical approaches such as the joint intentions, the joint responsibility, the SharedPlan concept)

Secondly, there is a question how to specify the policy that governs conversations. The main issue is what it means to understand or not to understand. Existing (AI) multi-agent systems tend to invest into conversation protocols based on the finite state transitions. This leaves us with fairly inflexible and simplistic conversational situation. Therefore, we ground the significant part of our work on the research that is done in the task based dialogue community with its commitment to common knowledge about tasks and agent roles.

2.4 Discussion

In this chapter, we considered the evolution of the speech acts from theoretical origins to the fully fledged agent communication language standards. This evolution is intertwined with the development of the theory of intention. This is a natural consequence of the recognition that a communication action, just as any other action, has an intention behind it and can be planned. In order to understand the speech act, a participant in a conversation has to understand the speaker's intention. It is important to note that the ACLs do not address the issue of resolving the speaker's intention. For example, the *inform* message type (from FIPA ACL) and its semantics do not determine if an actual *inform* message is really informing or if it actually motivates the hearer to perform an action and therefore functions as a *request*. The ACL's semantics are strongly relying on the agents' mental states. The mental state of the sender is defined as part of the conditions for sending a particular message, or as a result of sending a message. KQML additionally assumes that a hearer must be in a particular mental state to be able to process the messages. The agents' mental states are firmly grounded in the belief-desire-intention agency theory. The theory of intention and its descendant, BDI agency theory, model agents as entities with beliefs about the world, desires and intentions (chosen desires with commitment to achieve them). The BDI approach is at the core of the software agents' rational behavior.

Further research and analysis has shown that (speech act) intention recognition is far from a trivial effort. In a pursuit of a successful communication mechanism that overcomes this problem, the conversation protocol based on the finite state transition network was proposed. Many multi-agent systems base their communication abilities on the conversation protocols (e.g. InfoSleuth and Retsina). They proved to be useful in many situations since the idea of a protocol is to set up the expectations and to define a set of legal next messages given some current state in the protocol. But the state transition protocols are offering fairly inflexible and simplistic conversational structure.

A different approach was taken by the researchers investigating human-human or human-computer agent communication, and a more complex discourse structure was introduced. More complex mechanisms were proposed, such as understanding a speech act as a part of a discourse structure (the theory of discourse structure, SharedPlan) or as a planned action (the plan-based speech act theory). In addition, the system for delimiting possible intentions on the communication 'wire' was devised and it supported mixed-initiative dialogue between human and computer agent (Abstract Task Model). Some of the systems have developed models of the

shared task knowledge to intricate details (e.g. TAEMS). But the primary focus in that work has not been the communication action understanding.

In conclusion, the efforts in the agent communication and cooperation area have gone a long way, but there is still ground to be covered. Among the unresolved issues is the comprehensive conversation policy that exceeds the scope of the existing conversation protocols, and enables structuring and understanding of the speech acts and their content. Another open issue that naturally follows the previous one is the question of the contextual knowledge that is needed to ground the conversation policy. Elio et al. (2000a) proposed to ground the conversation policy in the knowledge about things such as the abstract joint task and the task interdependencies. The goal of our research is to have a structured approach to representing the task knowledge that allows messages to be resolved in the real time. We strive to develop a principled way to constrain the message content and the message exchange patterns. Our structured approach to the task knowledge is based on the XML representation. We are committed to a loose kind of the BDI agency, in the same sense that FIPA ACL assumes that BDI-like agents will use its messages. It means that as long as the connection can be established between the agents' internal data and processes with the mental states and attitudes assumed by the communication messages and interaction patterns, it is not relevant how those mental states are represented and implemented.

An open issue that emerges from this work concerns what it means for an utterance, speech act or agent message to be "not-understood". Recall the structure of this FIPA ACL message: *(not-understood :sender i :receiver j :content c)*, where *i* and *j* are agents, and *not-understood* message is i's response to j in the context of some previous message from *j* to *i*. The message content *c* is defined as a tuple consisting of an action or event (e.g., a just-received inform message from *j* to *i*) and a 'reason'. The occasion for sending *not-understood* is described as follows:

> The sender of the *not-understood* communicative act received a communicative act that it did not understand. There may be several reasons for this: agent may not have been designed to process a certain act or class of acts, or it may have been expecting a different message. For example, it may have been strictly following a pre-defined protocol, in which the possible message sequences are predetermined. The *not-understood* message indicates to the sender of the original (misunderstood) action that nothing has been done as a result of the message.... The second element of the message content tuple is a proposition representing the reason for the failure to

> understand. There is no guarantee that the reason is represented in a way that the receiving agent will understand. However, a cooperative agent will attempt to explain the misunderstanding constructively. (FIPA, 2003)

In the simple case, an agent would be justified in sending a *not-understood* message if it received a message that was not part of a predefined protocol, i.e., an illegal transition. For example, a *query* is not a legal transition from any of the nodes in Figure 2.6. Thus, an agent receiving a query could respond with *not-understood* and (possibly) construct a reason that corresponds to the assertion "your query is not a legal message for me to receive at this time."

In this research, we focus on principled and structured approach to defining the conditions and reasons for *not-understood* type of messages, which goes beyond this sort of syntactic protocol violation. We take a more task centered approach, borrowing from many of the themes and theories introduced here. This research presented in the remaining chapters proposes the notion of an agent interaction model for resolving the above issues. From this model, we define "error conditions" that constitute the occasions for sending the *not-understood* messages, the *reasons* for not understanding, and possible responses to it. Thus, interaction model subsumes the simple case of protocol violation and allows for a richer description of what legal, expected, and plausible messages are, based on a task model and agent roles within that model.

Chapter 3 Agents and Interaction Model

The main goal of our research is to investigate and propose the structure of an agent interaction model which can define more broadly the notion of *not-understood*. As presented in detail in the previous chapter, the communication and cooperation research efforts have generally settled on the idea that the communication is the key component of the cooperative, joint problem solving efforts (Cohen and Levesque, 1991; Jennings, 1995; Grosz and Kraus, 1996). It was suggested that the agents that are working on a joint task have to talk about *something* and that *something* is the progress of their task and other task related topics on the abstract and domain independent level (e.g., Elio and Haddadi, 1999). We base our research on these notions, and we investigate the ways to ground the agent conversation policies in the context of the shared task model, and to enable the run-time resolution of a message intention and meaning. Our proposed solution is the conceptual definition and computational implementation of what we call an *agent interaction model*.

One purpose of an interaction model is to define the space of possible intentions that cooperating and coordinating agents may have. The second purpose is to define the occasions on which it is expected or likely that those intentions would occur. The third purpose, related to the first two, is to delimit the message content, i.e., what appears in the *:content* field of a particular message. This requires more than just specifying a common content language and even a common ontology, since these alone do not delimit what agents can talk about.

An interaction model consists from the three main components that support the run-time message generation and resolution, and generation of the *not-understood* messages. The first component is a language that describes tasks and interdependencies among tasks, and hence interdependencies among agents assigned to those tasks. TAEMS (Decker and Lesser 1993; Decker et al. 1996; Decker 1996; Lesser 1998; Wagner et al. 1999; Lesser et al. 1999) is an example of such a language and we adopted a subset of its distinctions. The second component is a set of axioms and inference rules that agents can use, together with a model of task dependencies, to derive the initial and run-time propositional beliefs about other agents and other tasks. These are the propositions that agents will exchange, update, or revise through communication actions during a problem solving process. Particular exchanges, revisions, and updates are then understood as required, expected or plausible, *given* these task and agent models. The third component is concerned with the definitions of particular predicate classes, that define what kinds of the message content can be exchanged under various circumstances.

These interaction model components can be looked at as the *meat* for the speech act skeleton structure, combined together they provide the principles for the communication model that generates and resolves messages in the run-time. A *not-understood* message is generated as the answer to the received message that was outside of this model's definition of required, expected, or plausible messages.

The remainder of this chapter presents the details of an interaction model, namely the task structure modeling language and related beliefs, the structure behind the speech acts (and a *not-understood* message type in particular), and the axioms and inference rules for agent reasoning. These elements allow us to define the conditions under which an agent sends a not-understood message. Recall how FIPA describes this message type:

> The sender of the *not-understood* communicative act received a communicative act that it did not understand. There may be several reasons for this: agent may not have been designed to process a certain act or class of acts, or it may have been expecting a different message. For example, it may have been strictly following a pre-defined protocol, in which the possible message sequences are predetermined. The *not-understood* message indicates to the sender of the original (misunderstood) action that nothing has been done as a result of the message.... The second element of the message content tuple is a proposition representing the reason for the failure to understand. (FIPA, 2003)

Under this view, a *not-understood* messages constitute a kind of 'run-time error' that signals mismatches between agents' respective belief states, in terms of the general interaction model that defines legal and pragmatic communication actions. This research also considers the matter of how the interaction model may set policies for belief revision as a response to a *not-understood* message.

This chapter focuses on the conceptual aspects of the interaction model. The next chapter will re-visit these elements and provide more implementation details. This chapter starts by reviewing the assumptions made about the agents and their functionality. Next, it presents the description of the interaction model elements, namely the task structure modeling language and related beliefs, the structures behind the speech acts and *not-understood* message type, the predicate classes as content ontology, and the axioms for agent reasoning. We first present the specific domain problem that we used both as an inspiration and as a testbed domain environment

for these ideas. This will help the reader understand the more abstract aspects of the interaction model, which is intended (in principle) to apply to a wide range of tasks.

3.1 A motivating example: Distributed Airline Travel Planning Problem

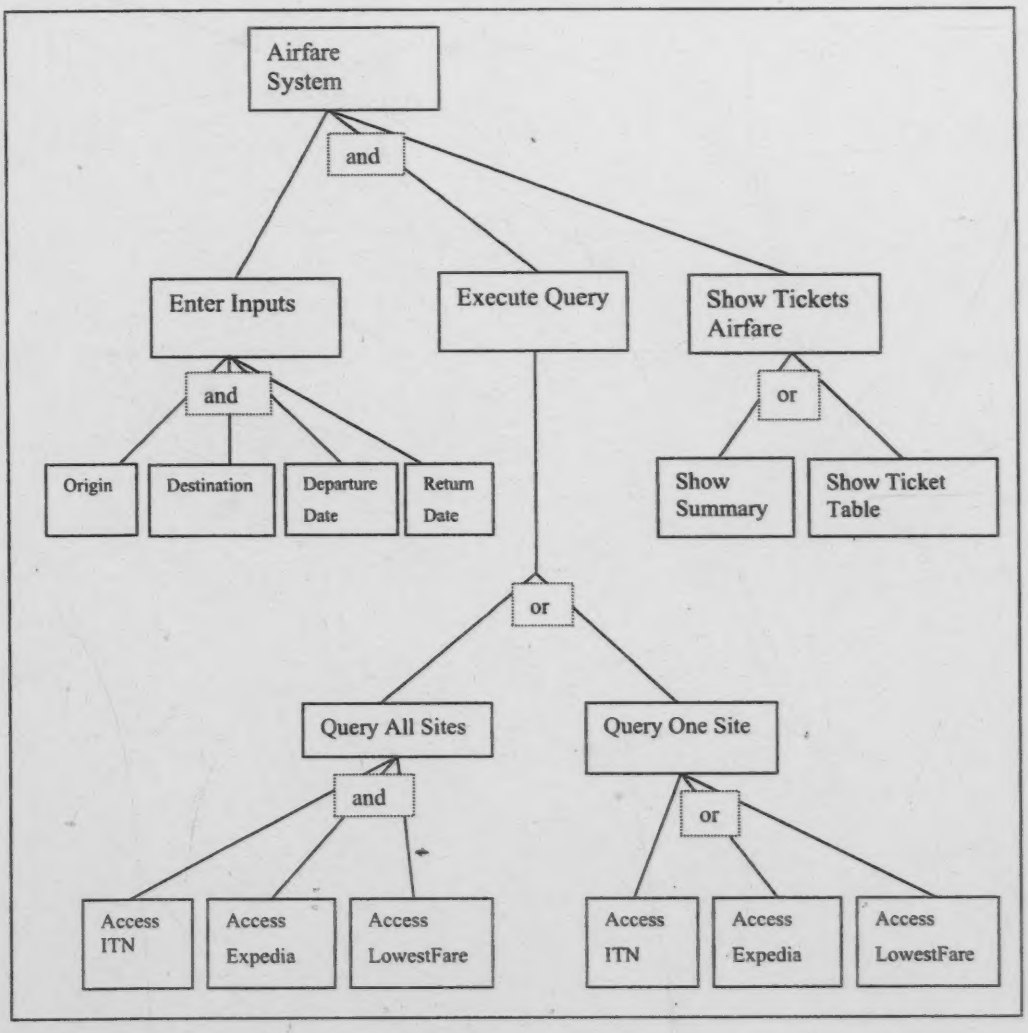


Figure 3.1 Task decomposition of the airline travel planning problem

We are interested in distributed multi-agent problem solving, and in particular, the communication elements that arise when tasks are distributed among independent problem-

solving entities. We present our defining features for what constitutes an 'agent' in a following section. For the moment, let us consider that they are some software entities that are given some problem specification and engage in some computation that, if given the proper resources, can bring about a solution to that problem. We assume a closed agent system: a single software developer develops all agents. The interaction model we develop in this chapter and the next is the means by which the agent designer defines, among other things, agent roles, responsibilities, and interdependencies. A key element is the specification of the overall task that is distributed as smaller tasks to a set of agents to solve.

Our example domain task is from the on-line airline reservation system domain. We have borrowed this general task specification from Situ and Stroulia (2000). Figure 3.1 shows a conceptual decomposition of this task.

The task presented in the Figure 3.1 is used mainly as a motivation and a concrete example of a task; it is more likely that this task would be handled by the group of heterogeneous agents; we are also aware that the above task does not necessarily require our solution but it gives us the opportunity to reason about the issues present in the larger scale multi-agent systems for distributed problem solving.

This kind of task decomposition implies certain requirements, i.e. the required input resources for each subtask. For example, an agent responsible for displaying the information about the found airplane tickets ('Show Tickets Airfare') needs this information as input resource (otherwise it is not able to successfully perform its task of displaying it). Based on the task decomposition, the agents have expectations about which other agents know what. For example, an agent working on the 'Execute Query' task expects its immediate parent to know the information about user's input, and it will expect to receive this information from it; it will not expect to receive this information from the agent working on 'Show Tickets Airfare' task since it will not expect this agent to know it. The execution of the subtasks belonging to the same parent task is determined with the decomposition. The 'and' decomposition requires all subtasks to be executed, and based on this, we can note that the information if one subtask fails is important to the parent agent, and that the parent agent knows that in this case the other subtasks have become irrelevant and it can wish to communicate this fact to them. Whether 'and' subtasks can be performed serially or in parallel depends on their interdependencies; for example, if subtasks are independent they can be executed in parallel, but if one requires resources generated by another or is enabled by another then they have to be executed in parallel. In the case of 'or' decomposition, parent agent needs only one subtask to be performed. Thus, it can choose one subtask and ask its agent for the results, but it could also ask all agents (responsible for the

subtasks) to execute them, then it can wait for the fastest results; once it gets them, it can inform other agents to stop (since their tasks are no longer relevant).

We have used this example to illustrate some of the important elements of how these agents would communicate effectively, efficiently, and intelligently, *given* some model of their responsibilities and roles in the overall task success. We have show by example some cases of what kinds of messages would be required or plausible to either send or receive. Messages that are not required or, more specifically, cannot be resolved within a model of what is required or plausible are messages that are *not understandable.* In remainder of this chapter, we refine what it means for messages to be defined and constrained by an interaction model, and hence the specific conditions under which a message is deemed not-understood. It is the different violations of the interaction model that will define the different reasons that a particular message is not understood. From this, we will be able to define quite specific "explanations" to put into the not-understood message, as per the FIPA vision for this message type.

Having given this concrete domain example, we will now move to an abstract version of a task and define both the interaction model and our assumptions about agents at this more abstract level.

3.2 What is an agent?

From the perspective of the agent interaction model, an agent is autonomous, problem solving software entity that is capable of high-level communication using FIPA ACL message types. An agent is defined by its belief state and its functional capabilities. We assume only that an agent has an internal belief state that consists of static and run-time beliefs. The static beliefs are defined by the interaction model, which we start to detail in the following section. An agent's functional capabilities are (a) the ability to solve the task to which it has been assigned, or determine it is not solvable, and (b) the abilility to generate and assimilate messages specified in a high-level ACL. Here, we sketch the functional assumptions we make about an agent at a very high level—the implementation of this functionality is the primary concern of Chapter 4.

3.2.1 Agent Functionality: Problem Solving

A task can be in one of these states: *not-attempted, possible, not-possible, irrelevant, failed,* or *succeeded.* An agent is the software entity that can transform the status of a task from its initial state (*not-attempted*) to a final state (*irrelevant, failed,* or *succeeded*). An agent has whatever domain level computational abilities necessary to do this. For example, the agent that is

responsible for checking the Travelocity site must have access to code that makes the connection to the site, searches it for the relevant information, and returns it. We are not interested in that level of specification, which can be done in any way that the agent developer finds to be most efficient. Instead, we are interested only in a high level ontology for describing task status and agent behavior related to task completion.

Considering this abstract problem-solving vocabulary, there must be some *glue* mechanism that connects domain specific computations with these more abstract state descriptors. In our implementation, we employ what we call the abstract task operators as part of that *glue*, i.e. an agent moves a task through the abstract problem solving states by applying the abstract task operators. An example abstract task operator (using English gloss) is:

If I intend task *t*, and status of task *t* is currently *not-attempted*
& I have all the resources and information specified for task *t's* initial state
& all semantic constraints on those preconditions are satisfied
then change task *t*'s status to *possible*

This operator specifies when problem solving on a task can begin, that is, the constraints that constitute its preconditions have been met and therefore progress on the task can begin. The task status is one of the more important propositions that agents reason and communicate about. These abstract operators can be considered as the agent's internal actions that are triggered by elements of its belief state. The complete set of abstract task operators are presented as part of the implementation details in Chapter 4.

3.2.2 Agent Functionality: Communication plans

Our agents are assumed to use FIPA ACL message. We are dealing with the *inform*, *disconfirm* and *query-ref* message types, but this model can be extended to accommodate other message types as needed. The generation of *inform*, *disconfirm* or *query-ref* messages within each agent is handled by a communication plan module. Communication plans are triggered in two ways. Firstly, they may be selected to enable a task operator, in a means-end manner; e.g., an agent may execute a *query* plan in order to obtain information about necessary resources to advance a task or a method, in other words, to satisfy a task or a method's preconditions and enable the transition from *not-attempted* to *possible* status. Secondly, the agents have communication plans to proactively generate *inform* or *disconfirm* messages for exchanging the

information believed to be useful to other agents. For example, an agent that determines that its own task has failed may inform the other agents who require its results or resources.

The communication plans for generating both kinds of communication correspond closely to the specifications of the FIPA ACL feasibility preconditions and rational effects, plus the additional conditions based on the KQML pragmatic semantics (Labrou and Finin, 1994; Labrou and Finin, 1997) and Searle's conditions (Searle, 1969). We give a more detailed account of the speech act underlying structure later in the Section 3.4.

An agent's communication module also creates the necessary data structures for maintaining conversations (multi-message exchanges) between itself and several agents. The incoming messages are either resolved against the existing conversation structures or recognized as initiating a new conversation. The communication model ultimately implements the notion of "understanding" a message by passing the message through a series of not-understood filters. These are defined in Section 3.4.

3.3 Modeling Task and Agent Interdependencies

The interaction model framework requires a declarative representation of a *task*. We model a task as an abstract specification of a problem to be solved that has a number of defining properties. The solution method for a task is specified as either a directly executable *method* or via the achievement of a set of *subtasks*. A (sub)task is related to another task through either an *and*-decomposition or an *or*-decomposition. The tasks may also be related to each other through an enables/enabled-by relationship: if task *t1* enables task *t2*, then task *t1* must be completed before task *t2* can begin. A *task-structure* specifies a hierarchical decomposition of a task into a set of subtasks whose leaf nodes are executable methods. As part of our modeling assumptions, each task and method in a task structure is assigned to exactly one agent, although one agent may be responsible for more than one task or method. Figure 3.2 presents a fragment of an (abstract) task structure to illustrate these properties.

Our task definition also includes preconditions and postconditions, which are used to define its initial state and its goal state. Preconditions and postconditions are stated as Boolean constraints on the domain-specific variables relevant to initiating a task and considering it successful, respectively. Any kind of a domain-dependent check can be specified in this way; we assume that agents have the domain-dependent code to determine if the constraint is satisfied. In our more abstract domain-independent language, the preconditions are regarded as resources that an agent requires in order to begin working on a task. Similarly, to declare that a task is

completed and successful, an agent must acquire beliefs that a task's results have been achieved (i.e., that the postcondition constraints are holding). It is during the course of satisfying such constraints that an agent moves its task from an initial (abstract) status state of *not-attempted*, through other abstract status states named *possible, not-possible, succeeded, failed*, and *irrelevant*.

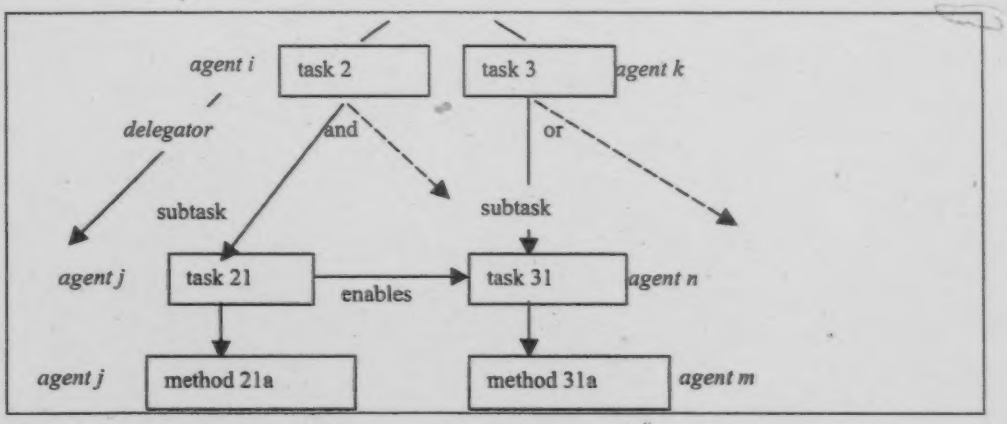


Figure 3.2 Elements of task structure and derived agent roles

Each agent has access to an entire task structure (the complete decomposition of a root task and the assignment of subtasks and methods to other agents), although this is not strictly necessary for our purposes. An agent's belief state is initialized and modified during run time by the agent's use of the shared task structure coupled with a set of axioms and inference rules. This is the second, major component of the interaction model, which serves to unite a model of task interdependencies with traditional belief-desire-intention perspective for modeling an agent's internal state.

The role of agent *i* to agent *j* in Figure 3.2 as *delegator* is derived from such inference rules applied to the jointly held task model. Similarly, the belief that agent *n* desires the results from task-21 is derived from beliefs that agent *n* is the agent assigned to that task, and that task 21 enables agent *n*'s task. These ultimately serve as the states that correspond to preconditions for communication acts. Table 3.1 shows a subset of the beliefs that would follow from this kind of inference rules, combined with the task structure in Figure 3.2.

beliefs about self	(agent-for j task-21) (relevant task-21) (intend j task-21)
	(agent-for j method-21a)
	(desire (j (result-for (task-21 result-a)))
	(enables task-21 task-31) (subtask-of task-21 task 2) ...
beliefs about others	(agent-for n task-31) (agent-for i task 2)
	(intend n task-31) (relevant task-31)
	(desire (i (result-for (task-21 result-b))))....
task progress	(relevant task-21) (status task-21 possible)
	(have-all-resources task-21) (valid-all-resources task-21)
	(decomposition task-21 method-21) ...
	(status method-21 not-attempted) ....

Table 3.1 Illustrative belief state elements for agent j (from Figure 3.2)

At the implementation level, we allow one level of nesting for the belief propositions, and drop the outer most *believe* when constructing an agent's beliefs about itself. *(Desire agent α)* is used to signify that an agent will aim to achieve a belief state in which proposition α is true. The *intend* predicate signifies an attitude towards a task by an agent; *intend(agent-i task-23)* is shorthand for signifying a behavioral commitment on agent *i*'s part to bring task-23 to some final state, whether that final state is *succeed, failed* or *irrelevant*. In using this kind of vocabulary, it is important to specify axioms to define the semantics of these concepts. We present these axioms in Section 3.3.2. From our perspective, the domain-specific computations on domain-specific information states can be carried out in any fashion, as long as they ultimately create beliefs in the agent that are represented using this abstract vocabulary element forms using this (or some other) abstraction vocabulary. The implementation is required to ensure that, during execution, agent belief states are internally consistent with the axioms defining the semantics. In the next section we enumerate the remaining predicates that are used to define agent beliefs.

3.3.1 Predicate and Predicate classes

We define a set of predicate classes and their properties. They specify the possible types of the message content that can be exchanged, and in which circumstance it can be exchanged. These predicates also present the agent mental state attitudes, that can be implemented in the explicit or implicit form.

Predicates	Predicate properties				
	Changeable or Fixed	*Public or Private*	*External or Internal*	*Negation form (explicit negation form, negation by failure*	*Comment and predicate proposition form*
agent	fixed	public	N/A (internal)	negation by failure	agent(a, t): Agent *a* is responsible for task *t*.
able	changeable	private	internal	negation by failure	able(a, t): Agent a is capable of performing task *t*.
intend	changeable	public	internal	negation by failure	intend(a, t): Agent *a* intends to bring task *t* to a final state.
desire	changeable	public	internal	negation by failure	desire(a, α): Agent *a* desires to achieve state where α is true.
know	changeable	public	internal	negation by failure	know(a, α): Agent *a* believes that α is true.
relevancy	changeable	public	external	negation by failure	relevant(t): Task *t* is to be performed as part of the current task-structure.
done	changeable	private	internal	negation by failure	done(t): Task *t* is in the final state and its post processing is finished.
status	changeable	public	internal	N/A	status(t)=not-attempetd: Task t has status 'not-attempted'.
resources	changeable	public	external	N/A	resource(r, 'xy'): Resource *r* has assigned value 'xy'.

results	changeable	public	internal	N/A	result(r, 'xy'): Result *r* has assigned value 'xy'.
external/ internal	fixed	public	N/A (internal)	negation by failure	external(p): Predicate class *p* can be externally revised.
public/ private	fixed	public	N/A (internal)	negation by failure	public(p): Predicate class *p* is public, and it can appear as a content proposition.
changeable / fixed	fixed	public	N/A (internal)	negation by failure	changeable(p): Predicate class *p* can be revised.
internal book-keeping beliefs	changeable	private	internal	explicit negation form	An agent needs beliefs that keep track of the problem solving process such as: valid-all-resources(t), it states that an agent's resources for task *t* are semantically valid

Table 3.2 A set of the predicate classes

Table 3.2 shows the list of the predicates along with their predicate classes and properties. The private/public property determines the permission of sharing beliefs, i.e. it distinguishes the propositions that *may be* exchanged within a message content from those propositions which *must not be* exchanged within a message content. For example, agents responsible for 'Enter Inputs' and 'Origin' tasks (Figure 3.1) may exchange messages about *results*, but they may not exchange messages with *have-resource* predicate since it belongs to the *private* set of predicates (*internal book-keeping beliefs* in Table 3.3).

The changeable/fixed property specifies whether the proposition can be revised (in case it is changeable) or if it cannot be revised (in case it is fixed). Essentially, this specifies whether a particular belief is defeasible. For example, a belief based on predicate *agent* cannot be changed (assuming that agents' responsibilities are static in the interaction model instance); the belief

about *status* predicate can be changed (since it reflects the dynamic, problem solving state of a task).

For defeasible or changeable propositions, the interaction model further specifies what the source of the belief update or revision can be. That is captured in the internal/external property. If the predicate is external then the other agents can update beliefs about it through the communication acts. If the predicate is internal then other agents have no right to change the agent's beliefs about this predicate. The above mentioned predicate *status* cannot be changed through the communication act (only the agent responsible for the task can update its status); on the other hand, predicate *resource* can be updated via communication act since, for example, a delegator agent is in many situations expected to update its delegatee's knowledge about task resources.

Each of the predicates has a negative form, most commonly it is 'negation by failure' form, but there is a subset of predicates that have explicit negative form. This subset of predicates is referred to as agent's *internal bookkeeping* beliefs because it is used by the agents to represent the state of the internal problem solving processes. The internal bookkeeping beliefs are presented in Table 3.3.

Internal bookkeeping predicate	Meaning
have-resource	It indicates availability of a resource needed for a task.
have-all-resources	It indicates availability of all resources needed for a task.
valid-all-resources	It indicates semantical validity of all resources needed for a task.
have-result	It indicates availability of a task result.
have-all-results	It indicates availability of all task results.
valid-all-results	It indicates semantical validity of task result.

Table 3.3 Internal bookkeeping beliefs

It is important to understand that the interaction model does not require that a particular predicate be in particular classes. We are arguing only that these predicate classes are important for constraining expected, plausible, and forbidden message content. Thus, in our specific examples, we classify *intend* as a public predicate. Under a different model, *intend* might be a private predicate. We simply offer an instance example of the property values which can be modeled and adjusted as needed.

3.3.2 Axioms and Inference Rules for Agent reasoning

Much has been written about the semantics for belief, desire, and intention. Our concerns in this work are more pragmatic. However, it is important to specify some minimal semantics for the predicates we have used in our own model, because agent behavior and reasoning must be driven by such semantics. We present these axioms relying on the above described predicates, they are given in the English form and the mathematical notation.

A1)Every task *t* has an assigned agent *a* and only one such agent that is responsible for it.

$\forall t \; \exists! a \; s.t. \; agent(a, t)$

A2)If an agent *a* is assigned to a task *t* and *t*'s status is not *failed*, then *a* is able to perform *t*.

$(agent(a,t) \wedge status(t) \neq failed) \Rightarrow able(a,t)$

A3)If an agent *a* is responsible for a task *t* and it has a motivation to do it, then the task is relevant.

$(agent(a,t) \wedge m) \Rightarrow relevant(t)$

We could more formally state that an agent is motivated to perform task *t*, i.e., motivated(a, t). What this axiom aims to capture is that there must be some other 'reason' to for a task to become relevant other than merely having an agent assigned to it. This reason can come from the outside, e.g. another agent requests the results for task *t*, or from inside the agent, e.g. agent can infer that its task is relevant because it is a top level task. The belief that a task is relevant is crucial, because it drives all subsequent problem solving and communication behavior.

A4)If the agent *a* is responsible for task *t*, then it knows the status of *t*.

$agent(a, t) \Rightarrow know(a, status(t), t)$

This axiom is important since it allows agents to infer which agents know the status of which tasks based on their responsibilities.

A5)If the agent *a* is responsible for task *t*, *t* is relevant, and *t*'s status is not *succeeded* or *failed*, then *a* intends to perform *t*.

$$(agent(a,t) \wedge relevant(t) \wedge (status(t) \notin \{succeeded, failed\})) \Rightarrow intend(a,t)$$

A6)Only the agent *a* assigned to a task *t* can intend it.

$$intend(a,t) \Rightarrow agent(a,t)$$

A7)The agent *a* desires to know resources *r* for the task *t* or already knows it, iff *a* intends to perform *t*.

$$(desire(a, know(a, r, t)) \vee know(a, r, t)) \Leftrightarrow intend(a, t)$$

This axiom is important because an agent can make inferences about other agents that intend to perform their tasks, e.g. if agent believes that other agent *a2* intends task *t* and that it does not know the resources for it, then it can infer that *a2* desires those resources.

A8)The agent *a* desires to know the result *r* for the task *t1* or knows it, iff *a* is responsible for the task *t1*, it intends to perform *t1* and *t1* is not in the final state where final state is defined as Final_state $\in$ {*failed, succeeded*}.

$$desire(a, know(a, r, t1)) \Leftrightarrow (agent(a, t1) \wedge intend(a, t1) \wedge \neg know(a, r, t1))$$

This axiom provides the agent with possibility to infer that its delegator agent (or enablee agent) desire to know the results of its task.

A9)If the agent *a* is responsible for task *t*, and *t*'s status is *successful*, then *a* knows *t*'s results.

$$(agent(a, t) \wedge (status(t) = successful)) \Rightarrow know(a, result(t), t)$$

We are not committed to axiomatize the complete interaction model. We are prescribing the axioms that define the predicate semantics. For example, we expect from the agent

architecture, if using this interaction model, to implement behavior that is consistent with the semantics associated with 'intend', 'know', and so forth.

3.4 Assimilating Speech Acts: Occasions for not-understood

So far in this chapter, we have presented the key elements of an interaction model, namely the means for describing a distributed task structure, for assigning agents to portions of that distributed task structure, and for defining and inferring the propositions that would define an agent's belief state. This is the infrastructure that an agent uses to generate required, expected, or plausible messages *and* for assimilating messages it receives from other agents. In this work, we are concerned only with *inform, query-ref, disconfirm*, and *not-understood* (these speech acts are presented in Section 2.2.2).

In this section, we present the general intuitions that define how we use this infrastructure to define illegal communication, namely the conditions under which an agent sends a *not-understood* message, and the reasons it formulates to accompany the message.

To illustrate our approach, Table 3.4 presents a general schema for an *inform* message type, which takes as its content a proposition *p*. Our approach towards understanding and 'not understanding' a message and the related ontology starts by investigating agent *j*'s inability to explain how one or more of these conditions could hold for agent *i* as the sender, given that agents *i* and *j* jointly share this general schema for an *inform* action and a common interaction model.

<inform i, j, p>	
feasibility preconditions:	*i* knows *p*
	i believes *j* has no position on *p*
contextual relevance:	*i* believes *j* desires to know *p*
intended effect	*j* knows *p*
success conditions:	*p* is interpretable to *j*
	j's mental state concerning *p* can be updated
	j's mental state concerning *p* can be updated by *i*

<inform i, j, p > is not-understandable to *j* wrt an interaction model *m* if by that model

i) *p* 's predicate is not in a commonly shared ontology

the intuition: "I don't know what you are talking about."

model component: public vs. private predicates

ii) *j*'s position on *p* cannot be updated/revised

the intution: "Beliefs about *p* cannot be changed."

model component: fixed vs. changeable predicates

iii) *j*'s position on *p* can only be updated/revised by *j*'s own reasoning, or if *i* holds a particular role to *j*

the intuition: "*You* cannot change my belief about *p*"

model component: external vs. internally changeable predicates; agent roles derived from shared task model

iv) *j* cannot explain why *i* would know *p*

the intuition: "How can *you* know *p*?"

model component: agent acquaintance model derived from shared task model

v) *j* cannot explain why *i* believes *j* has no position on *p*.

the intuition: "Why do you believe I do not already *know p*?"

model component: agent acquaintance model

vi) *j* does not desire to know *p*

the intuition: "*Why* are you telling me *p*? It would have no impact on my behavior."

model component: agent acquaintance models derived from shared task model

---

Table 3.4 The schema for the *inform* message type and six categories of *not-understood*

The lower part of Table 3.4 illustrates six categories of *not-understood* with some accompanying intuitions. The first three are concerned with the propositional content of the message itself, the particular illocutionary force applied to the content, and the role of the sender to the receiver. Case (i) is a simple matter of whether the predicate of proposition *p* is in the shared ontology of the agents. Such predicates are classed as *public* and are allowable in the propositional content of a message. The interaction model designates *private* predicates as those used to construct belief state propositions that might be idiosyncratic to particular agents, or that might be restricted from sharing. Case (ii) stipulates what propositions involving public

predicates are defeasible, and which are not, by classifying predicates as *fixed* or *changeable*. Case (iii) covers the matter of whether a defeasible belief of agent *j* is revised only through *internal* reasoning actions or via an *external* communicative act of another agent. For example, an interaction model might specify that agent *i* may not *inform* agent *j* what agent *j* believes, intends or desires, to embody the constraint that revisions to *j*'s mental attitudes are the province of agent *j*. (These cases loosely correspond to canonical examples such as "I insult you." "I convince you of p."(Austin, 1962)). But we extend this distinction to other types propositions in *j*'s belief state, namely any proposition for which only *j* can determine a truth status.

Cases (iv) through (vi) focus on *not-understanding* as violations of some initial or derived 'acquaintance model' of other agents, i.e., beliefs about other agents, their capabilities, their intentions, and so forth. The systems of coordinating or cooperating agents often implicitly or explicitly rely on such acquaintance models, derived from a global or partial model of interdependencies among tasks and the agents assigned to those tasks (Grosz and Sidner, 1990; Lochbaum et al., 1990; Decker and Lesser, 1995). In these cases, a *not-understood* message with a 'constructive' reason signals a mismatch between agent *i* and *j*'s respective acquaintance models of each other. For example, cases (iv) and (v) indicate *j*'s inability to explain why—in terms of its beliefs about agent *i* and what agent *i* ought to know about agent *j*— the feasibility conditions for *inform* are holding for *i*. Finally, case (vi) covers an important pragmatic case, from the viewpoint of a message's extended perlocutionary effects. Presumably, *i* intends that *j* adopt belief *p* so that *j*'s behavior will change. If belief in *p* would not impact any of *j*'s behaviors, there is no consequence of adopting it. In this instance, *j*'s *not-understood* signals a mismatch between *j*'s own model of contextual relevance, and agent *i*'s model of what is contextually relevant to *j*.

Table 3.5 shows the structure of the *disconfirm* speech act which is almost the same as the *inform* structure.

<disconfirm i, j, p>	
feasibility preconditions:	*i* knows $\neg p$
	i believes *j* knows *p*
contextual relevance:	*i* believes *j* desires to know *p* (or $\neg p$)
intended effect	*j* knows $\neg p$
success conditions:	*p* is interpretable to *j*
	j's mental state concerning *p* can be updated
	j's mental state concerning *p* can be updated by *i*

< *disconfirm i, j, p*> is not-understandable to *j* wrt an interaction model *m* if by that model

i) *p* 's predicate is not in a commonly shared ontology

the intuition:	"I don't know what you are talking about."
model component:	public vs. private predicates

ii) *j*'s position on *p* cannot be updated/revised

the intution:	"Beliefs about *p* cannot be changed."
model component:	fixed vs. changeable predicates

iii) *j*'s position on *p* can only be updated/revised by *j*'s own reasoning, or if *i* holds a particular role to *j*

the intuition:	"*You* cannot change my belief about *p*"
model component:	external vs. internally changeable predicates; agent roles derived from shared task model

iv) *j* cannot explain why *i* would know $\neg p$

the intuition:	"How can *you* know $\neg p$?"
model component:	agent acquaintance model derived from shared task model

v) *j* cannot explain why *i* believes *j* knows *p*.

the intuition:	"Why do you believe I do already *know p*?"
model component:	agent acquaintance model

vi) *j* does not desire to know *p*

the intuition:	"*Why* are you telling me *p*? It would have no impact on my behavior."
model component:	agent acquaintance models derived from shared task model

---

Table 3.5 The schema for the *disconfirm* message tupe and six categories of *not-understood*

Table 3.6 presents the structure of the *query-ref* communication action and the occasions for *not-understood*. In the same manner as with the *inform and disconfirm* case, it explores the legality of its propositional content in cases i), ii) and iii). Cases iv), v) and vi) explore the communicational violations based on the initial or derived beliefs about other agents. The iv) and v) cases question the feasibility preconditions for the sender *i*. The vi) case investigates the

meaningfulness of the extended perlocutionary effect, i.e. it questions whether answering the query would make sense based on the agent acquaintance model.

<query i, j, p>

feasibility preconditions:	*i* has no position on *p*
	i believes *j* knows *p*
contextual relevance:	*i* desires to know *p*
intended effect	*i* knows *p*
success conditions:	*p* is interpretable to *j*
	j's mental state concerning *p* can be accessed
	j's mental state concerning *p* can be accessed by *i*

<query i, j, p> is not-understandable to *j* wrt an interaction model *m* if by that model

i) *p* 's predicate is not in a commonly shared ontology

the intuition:	"I don't know what you are talking about."
model component:	public vs. private predicates

ii) *j* believes *i*'s position on *p* cannot be updated/revised

the intution:	"Beliefs about *p* cannot be changed."
model component:	fixed vs. changeable predicates

iii) *j* believes *i*'s position on *p* cannot be updated/revised by *j*

the intuition:	"I believe *I* cannot change your belief about *p*"
model component:	external vs. internally changeable predicates; agent roles derived from shared task model

iv) *j* cannot explain why *i* would not know *p*

the intuition:	"How come *you* do not know *p*?"
model component:	agent acquaintance model derived from shared task model

v) *j* cannot explain why *i* believes *j* knows *p*.

the intuition:	"Why do you believe *I know p*?"
model component:	agent acquaintance model

vi) *j* does not believe that *i* desires to know *p*

the intuition:	"*Why* are you asking me *p*? It would have no impact on your behavior."

model component:	agent acquaintance models derived from shared task model

Table 3.6 The schema for the query and six categories of *not-understood*

We wish to make a few additional points about the three schemas presented above. We have not explicitly included the canonical *not-understood* case of a protocol violation for two reasons. First, a predefined message exchange protocol can be viewed as a special instance of case (vi) with explicit reference to the context in which it was received (e.g., "an *inform* of any sort at this point has no impact on my behavior"). We readily acknowledge that such violations are more easily recognized as syntactic violations, but that is a matter of processing convenience. More generally, we prefer a scheme whereby an agent recognizes required or plausible messages through appeal to a jointly shared task model that tracks task progress (Elio and Haddadi, 2000) and not via a predefined protocol. The shared task model is the cornerstone of the general interaction model.

Second, we have used the notion of 'explain' in Table 3.4, Table 3.5 and Table 3.6 to emphasize that a received message is to be reconciled against an interaction model. We have no particular investment in how simple or complex this explanation process might be nor do we expect that agents will necessarily discover a discrepancy by means of some theorem proving procedure applied to formulas in the semantic language that represents their mental states. Pragmatically, the interaction model need only specify what the language of these mental attitudes and abstract task predicates might be.

Finally, it is natural at this point to consider whether agent *i*, having received a *not-understood* message from agent *j*, would itself generate a *not-understood* message back to agent *j*, and so on. This can be a difficult issue, depending on how it is approached. We propose simple belief revision through appealing again to the interaction model; this will be discussed in detail in later chapters.

3.5 Summary

In summary, the approach we advocate is called the agent interaction model and its specifications define the foundations for constraining and understanding the meaning of the

messages being exchanged between agents in the context of their cooperative problem solving efforts.

An interaction model consists of at least these three components. The first component is some abstract shared task model that specifies how tasks and subtasks (to which agents are assigned) are interrelated. The second component is a specification of the kinds of beliefs that are allowable as the message content, that are changeable during the course of a problem-solving process, and that are changeable via a communication action (i.e. externally changeable). The third is a specification of the agent roles in the context of these interacting tasks, roles that effectively specify conventions according to which agents may revise, update, or access the belief states of other agents. Additionally, the axioms and inference rules constrain more the relations between task properties, beliefs and agent roles.

The structured approach to the message understanding naturally leads to the structured approach towards the message *not understanding* and reasons for it. The ontology we develop for the purpose of specifying propostional content also has the purpose and capability to specify the *communication error conditions* for each case of *not understanding*. This approach is explored through specifying the structures behind the speech acts and it identifies possible points of *not understanding*.

In the next chapter, we present the multi-agent system and agent architecture implementation with an interaction model used by agents that are cooperatively working on the simple airline search/reservation type of a task.

Chapter 4 Interaction Model and Multi-agent system Implementation

In the previous chapter we presented an agent interaction model and its components. The first component is the task modeling language for task structure representation including the agent roles and interrelationships. The second one is the classification of agent beliefs that can be used as part of propositional content of a message, and belief properties that specify their content related behavior. The third component consists of axioms and inference rules that semantically intertwine the elements of the first two components.

In this chapter we present the implementation of the multi-agent system and the interaction model. We incorporate the implementation of an interaction model in our system, and we describe the implementation modules that reflect the interaction model components. The interaction model described in the previous chapter presents the components and the principles while the instantiation of a model into an actual interaction system is left to a designer. In our system implementation we instantiate one possible interaction model but we stress that it can be changed and extended as needed by the problem solving situation.

In the first part of this chapter, we look into the implementation of the interaction model components, namely task structure ontology, communication related ontology and the interaction model synthesis form for communication.

In the second part, we look into the multi-agent system and agent architecture implementation. Firstly, we look into the functionalities of the simple multi-agent system framework that contains the agents. Secondly, we look into the agent architecture, its components and its implementation. An agent goes through the execution cycles that involve communication handling phase and operator execution phase. The abstract task operators and the task supporting operators lead an agent through the problem solving process, and incorporate the axioms from the interaction model. The communication plans are responsible for handling the conversation, i.e. for generating and understanding the messages. They are based on the interaction model speech act structure and the context sensitive grammar representation of the legal and pragmatic messages as presented in the section about the interaction model synthesis. Finally, we take a look into the task domain abilities of an agent, where the domain is the travel planning type of a task.

Figure 4.1 shows the schema of the agent architecture with its components and relationships between them. We based our implementation on the object-oriented programming language, Java, and the declarative forms of the interaction model are represented in XML (with

the corresponding DTD). Each of the components of the agent architecture shown in Figure 4.1 consists of Java classes.

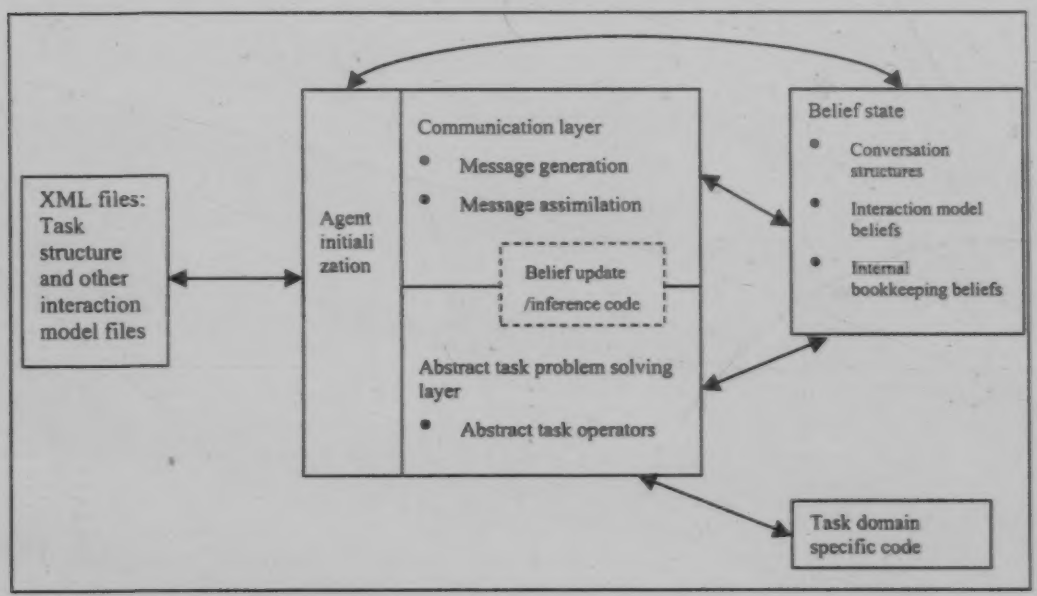


Figure 4.1 Agent architecture diagram

4.1 Agent Interaction Model Implementation

4.1.1 Task structure ontology

This section presents the ontology we use for task modeling. The resulting task structure is a tree-like form with task and method components as the nodes. The non-leaf nodes are denoting tasks, and the leaf nodes are referring to methods. The tasks and methods are defined as ontological classes with corresponding properties. During the system run-time they are instantiated as the task and method objects in the belief set of an agent.

Class: Task
Properties:

a) id
b) name
c) component_of (task_structure_id)
d) parent_list
e) children_list

f) enabled_by
g) enables
h) agent
i) Class: Decomposition
  Properties:
    1) child_type, oneOf: method, task
    if child_type=task
      2) cardinality: >0
      3) choice, oneOf: and, or
    if child_type=method
      2) cardinality, oneOf: =1
j) Precondition (instanceOf Constraint)
k) Postcondition (instanceOf Constraint)

Class: Constraint
Properties:
  a) id
  b) expression

Class: Variable
Properties:
  a) id
  b) data_type, oneOf: string, num, date, object

Table 4.1 *Task*, *Constraint* and *Variable* classes as part of the task structure ontology

The Task is a class representing atomic problem solving unit, i.e. the smallest task that can be assigned to an agent in order to be performed. It has a number of properties, some of them are classes themselves with their own properties. In case a property has 'oneOf' constraint on its value, only one of the specified values can be assigned to this property. Let us list and describe these properties (shown in Table 4.1).

The 'id' property is the unique identifier of a task. 'Name' is the human readable name of a task. The 'component_of' property connects a task to the task structure identifier, where identifier of the top-level task node in the structure is also the identifier of the whole task structure. The following two properties anchor a task in its position in the task structure, namely 'parent_list' (a list of parent tasks) and 'children_list' (a list of children tasks). The 'parent_list' and 'children_list' are the inverse properties. The 'agent' property points to the agent responsible for the task. We assume that the agent responsible for the task has the ability to check if the task preconditions and postconditions are satisfied.

The properties 'enables' and 'enabled_by' determine dependency relations between tasks where the *enabler* task has to finish successfully for the *enablee* task to be performed.

The 'decomposition' is a structurally crucial property for it provides the specification of a task's decomposition or execution. The 'child_type' property specifies whether child components are methods or tasks. Depending on the value assigned to the 'child_type' property, the different sets of properties will be required, as described in the following paragraph.

If the 'child_type' property has the value *task*, then 'cardinality' and 'choice' properties are required. The 'cardinality' property determines the number of children and in this case it is constrained to the whole number larger than zero. The property 'choice' denotes that a task can be decomposed into the children tasks that can be executed in two different manners. Firstly, the children tasks can be executed with the 'and' constraint, i.e. all children tasks have to be executed. Secondly, the children tasks can be executed with the 'or' constraint, which means that only one of the subtasks has to be chosen and (successfully) performed. If the 'child_type' has value *method*, then the only other property that is relevant is 'cardinality', and it has value '1' by default.

The next set of properties of the Task class is called Constraints. The Preconditions and Postconditions are instances of the class Constraint. The Preconditions determine what conditions have to be satisfied before a task execution can begin. The Postconditions determine the conditions that have to be satisfied for a task to be successfully concluded. Each Constraint instantiation, which can represent a task's preconditions or postconditions, consists of the 'id' and 'expression' properties. They denote a constraint identifier and its expression, i.e. a Boolean clause specifying constraints on the domain-specific variables. In our airline ticket reservation application, a simple precondition to the task of obtaining user information would include that the departure and return dates are unknown; a postcondition is that both such dates are known and that the former occurs before the latter. Any kind of a domain-specific check can be specified in this way; we assume that the agents have the domain-specific code to determine if the constraint is satisfied.

The Variable class provides the information about the variables that can be found in the constraint expressions in a task structure. It contains the variable identifier 'id' and 'data_type' that specifies if a variable is a string type, a number type, a date type or a generic object type.

Class: Method
Properties:

a) id
b) name
c) component_of (task_structure_id)
d) parent_list
e) enabled_by
f) enables

g) agent
h) Precondition (instanceOf Constraints)
i) Postcondition (instanceOf Constraints)
j) executable_code_path

Table 4.2 *Method* class as part of the task structure ontology

The Method is a class that represents the calls of the executable domain-specific code within the task structure (shown in Table 4.2). The Method instantiation represents a leaf node. Some of its properties have the same name and the same function as those belonging to the Task class. These are the following: 'component_of', 'id', 'name', 'parent_list', 'agent', 'enabled_by', 'enables', 'Precondition' and 'Postcondition'. The Method class has an additional property called 'executable_code_path' that specifies the path for accessing the executable domain-specific code.

Figure 4.2 shows the excerpt form the airline travel planning task structure in XML using the above representation (please note that we include regular characters instead of XML escape characters in the examples for the presentational purposes).

```
<Task>
        <property id="T021"/>
        <property name="Enter Input"/>
        <property component_of="T011"/>
        <property parent_list="T011"/>
        <property children_list="T031, T032"/>
        <property agent="ag3"/>
        <Decomposition>
                <property child_type="task"/>
                <property cardinality="2"/>
                <property choice="and"/>
        </Decomposition>
        <Postcondition>
                <property id="origin"/>
                <property expression="origin!=null"/>
        </Postcondition>
        <Postcondition>
```

```
                <property id="destination"/>
                <property expression="destination!=null"/>
        </Postcondition>
</Task>
<Method>
        <property id="M031"/>
        <property name="Origin"/>
        <property component_of="T011"/>
        <property parent_list="T031"/>
        <property agent="ag1"/>
        <Postcondition>
                <property id="origin"/>
                <property expression="origin!=null"/>
        </Postcondition>
        <property executable_code_path="test1method31"/>
</Method>
<Variable>
        <property id="origin"/>
        <property data_type="string"/>
</Variable>
```

Figure 4.2 Excerpt from a task structure in XML

This XML excerpt shows one task, one method and one variable encoded in the XML using the task structure ontology. In the task structure, such as shown in the Figure 3.2, each of those task and method nodes is represented in the declarative form as shown in the above XML excerpt. A Document Type Definition (DTD) file developed for defining and validating the structure of these XML files is shown in Appendix B.

4.1.2 Explicit Communication Object Ontology

The explicit communication object (ECO) ontology presents the ontology for the declarative interaction model knowledge that comprises the allowable predicates, the allowable

modalities, the allowable agent roles, and the allowable speech acts (or performatives) which has to be the subset of the FIPA ACL communicative act library.

The ECO contains few sections, namely the *predicate* section, the *modality* section, the *role* section and the *performative* section. The *predicate* section defines what agents can talk about, i.e. what can appear as the content of the message. It consists from the list of the predicates that are specified in the manner presented in the Table 3.2. On the ontology level, each predicate has an identifier and three properties, as defined in Chapter 3. Predicate section structure:

a) id
b) private/public
c) changeable/fixed
d) internal/external

The *modality* section lists possible agents' attitudes towards propositions. The *role* section lists possible agent roles in a task structure. The *performative* section lists allowable FIPA ACL communication acts.

```
<ExplicitCommunicationObjects>
    <predicate>
        <relevancy>
            <public/>
            <external/>
            <changeable/>
        </relevancy>
        <status>
            <public/>
            <internal/>
            <changeable/>
        </status>
        <agent>
            <public/>
            <internal/>
            <changeable/>
        </agent>
        [...]
```

```
</predicate>
<modality>
        <intend/>
        <know/>
        <desire/>
</modality>
<role>
        <parent/>
        <child/>
        <enabler/>
        <enablee/>
</role>
<performative>
        <query_ref/>
        <inform/>
        <disconfirm/>
        <not-understood>
</performative>
</ExplicitCommunicationObjects>
```

Figure 4.3 Explicit Communication Ontology in XML

The above example in Figure 4.3 shows the XML representation of the explicit communication object ontology instantiation as part of an interaction model declarative knowledge. A Document Type Definition (DTD) file that defines and validates the structure of this kind of XML files is shown in Appendix B.

4.1.3 Interaction Model Synthesis Form

An interaction model has a number of elements that can be used to enumerate the set of all messages that adhere to its constraints. It is convenient and useful to re-represent the interaction model elements as a context sensitive grammar which an agent can use to generate (or validate) messages. Figure 4.4 shows such a context sensitive grammar for generating schematic

versions of an *inform* and *disconfirm* message, created (by hand) from an interaction model specification:

inform-and-disconfirm-messages	⇒	msg-1 | msg-2
sa-1	⇒	inform | disconfirm
msg-1	⇒	(sa-1 sender receiver role-of$_{s\text{-}to\text{-}r}$ (task-subject))
task-subject	⇒	task-of$_{sender}$ | task-of$_{receiver}$
role-of$_{s\text{-}to\text{-}r}$	⇒	delegator | delegatee | enabler | enablee
external	⇒	relevant | resource
modality	⇒	intend | know | desire
[delegator] [task-subject]	⇒	[delegator] [task-subject] [external]
[delegatee] [task-of$_{sender}$]	⇒	[delegatee][task-of$_{sender}$][result]
[role-of$_{s\text{-}to\text{-}r}$][task-of$_{sender}$]	⇒	[role-of$_{s\text{-}to\text{-}r}$][task-of$_{sender}$][status]
msg-2	⇒	(sa-1 sender receiver role$_{s\text{-}to\text{-}r}$ (sender modality)

Figure 4.4 A context sensitive grammar for the legal *inform* and *disconfirm* messages allowable by an interaction model (*s-to-r = sender-to-receiver*)

There are a few points to make about this type of grammar for the interaction model. First, any agent in our system can run this grammar by first binding sender to itself and receiver to some particular other agent. *Task-of* is a function that takes a particular agent as an argument and returns a task to which that agent is assigned. Second, the grammar embodies constraints about public, changeable, and external predicates *as well as* pragmatic considerations about what constitute plausible *informs* and *disconfirms* from one agent to another. The latter are captured through reference to the role of the speaker to the receiver. This grammar generates schematic *informs* (and *disconfirms*) such as:

(inform sender receiver delegator (task-of$_{sender}$ relevant))

This embodies the interaction constraints that it is legal and *pragmatic* for an agent *i* (as sender) to inform agent *j* (as receiver) of agent *i*'s own task relevance, if agent *j* is performing a subtask for agent *i*. It is legal, because agent *i* can send messages with propositional content about task relevance (relevance is public) and relevance can be updated via communication acts (it is external). It is pragmatic, because agent *j* might infer its own task should be initiated (or stopped) by learning about agent *i*'s task relevance. This grammar would not generate:

X(inform sender receiver delegatee (task-of$_{receiver}$ relevant))

If the sender is a *delegatee* to the receiver, it cannot inform the receiver (the *delegator*) about the relevance of its task, it does not have the appropriate role to be allowed to do that.

For convenience, we define type-2 messages as having a slightly different structure, and the grammar would generate this as an instance:

(inform sender receiver delegatee (sender knows))

These type-2 informs simply specify that a sender can inform a receiver about the sender's attitudes towards propositions. However, this particular grammar is based on an interaction model that requires that *some* direct role exist between the sender and receiver for such a message (e.g., the grammar would disallow this message from agent *i* to agent *k* in Figure 3.2).

---

query-ref-messages	⇒	msg-3 | msg-4
msg-3	⇒	(query-ref sender receiver role-of$_{s\text{-}to\text{-}r}$ (task-subject))
task-subject	⇒	task-of$_{sender}$ | task-of$_{receiver}$
role-of$_{s\text{-}to\text{-}r}$	⇒	delegator | delegatee | enabler | enablee
external	⇒	relevant | resource
modality	⇒	intend | know | desire
[delegatee] [task-subject]	⇒	[delegatee] [task-subject] [external]
[delegator] [task-of$_{sender}$]	⇒	[delegator][task-of$_{receiver}$][result]

$[\text{role-of}_{\text{s-to-r}}][\text{task-of}_{\text{sender}}] \Rightarrow [\text{role-of}_{\text{s-to-r}}][\text{task-of}_{\text{receiver}}][\text{status}]$

$\text{msg-4} \Rightarrow (\text{query-ref sender receiver role}_{\text{s-to-r}} (\text{receiver modality})$

Figure 4.5 A context sensitive grammar for the legal *query-ref* messages allowable by an interaction model (*s-to-r = sender-to-receiver*)

Figure 4.5 presents the context sensitive grammar for the *query-ref* message type. This grammar represents the propositional constraints (public, changeable, etc.) and pragmatic constraints on the *query-ref* messages (based on the structure presented in Table 3.6). An example message generated by this grammar is:

(query-ref sender receiver delegator ($\text{task-of}_{\text{receiver}}$ result))

This shows a legal and pragmatic *query-ref* message. It is legal because a sender can send a querying message with the public propositional content (*result* predicate is public) that refers to the receiver's belief (*result* of the receiver's task). It is pragmatic because a delegator delegates a task in order to receive the results from that task. An example of a message that would not be generated by the above grammar is:

X(query-ref sender receiver delegatee ($\text{task-of}_{\text{sender}}$ result))

A sender cannot ask the receiver the results of the sender's task. It is based on the principle (encoded in an axiom) that if the agent is responsible for the task, then it knows its results (when the task is finished). Therefore, an agent can only internally generate the results of its own tasks.

We implement this type of grammars for *inform*, *disconfirm* and *query-ref* message types. This kind of re-representation of the interaction model is useful because it gives an agent the procedural capability of (a) generating all legal and pragmatic messages between itself and another agent and (b) checking whether the general form of an incoming message from an agent in a particular role passes various types of understandability filters. The grammars also serve an important function for us in the handling of not-understood message exchanges, namely to forbid certain kinds of replies *to* a *not-understood* message .

4.2 Agent Architecture and Multi-Agent System Implementation

4.2.1 Multi-agent system

The multi-agent system is implemented as a container framework that sequentially invokes agent entities. We adopt the simple approach towards the multi-agent system framework in order to 'push' the capabilities towards individual agents and make them autonomous entities.

Multi-agent system execution cycle:

Initialize *n* agents
While task structure *ts* is not done
    For each of *n* agents
        Invoke agent execution

Figure 4.6 Multi-agent system execution cycle

Figure 4.6 presents the framework execution cycle, where *n* is an arbitrary number of agents present in the system and jointly working on the task represented with the task structure *ts*. The *Invoke agent execution* statement refers to the execution of an agent cycle as presented in Figure 4.7.

4.2.2 Agent and Agent Execution Cycle

Generally speaking, an agent has beliefs about the world state and it has capabilities to perform the actions that change the world state. In order to explore the agent architecture in our system, we will talk in details about what an agent believes and what an agent does.

The initial agent beliefs are inferred from the declarative (XML) form of the task model. In the Section 4.1.1 we gave the detailed review of the task modeling language and the explicit communication object ontology that also serves as belief ontology. The agent initialization module (Figure 4.1) is responsible for the initial belief inference, and for the general initialization of the agent framework and its components. Agent belief state (Figure 4.1) contains initially inferred beliefs and beliefs dynamically acquired in the problem solving process.

Every agent in our multi-agent system is sequentially invoked (Figure 4.6). After being invoked by the system, each agent goes through its execution cycle as presented in the Figure 4.7 containing the pseudo-code of the agent execution cycle. Each execution cycle run invokes the communication module and the problem solving module in order to achieve progress on agent's tasks. Figure 4.1 shows that both communication module and problem solving module rely on agent's belief state to perform their respective processing tasks.

The first part *Process received messages* refers to the conversation handling module that invokes communication plans for message understanding (described below). The second part depicts the problem solving execution cycle. Firstly, for each task the agent is actively working on and based on the present state of the problem solving process, a set of the applicable operators is chosen. Then, an applicable operator that gets triggered is executed. In case no operator triggered, or the operator that triggered was the abstract task operator or the supporting operator that can possibly spawn a communication intention (as described below), then the agent cycle is done. If a triggered operator belongs to the supporting operators that performs only internal computations, then the agent continues with its cycle and operator execution.

An agent execution cycle:

Process received messages
    Try to understand received messages and/or to complete the open conversations
    Belief update
Operator processing
    For each activated task
        For each applicable operator for the task, try matching operator's trigger with the agent's beliefs
            If an operator triggered
            Then execute the operator
                (Possible communication intention spawning)
            If triggered operator is task operator or supporting task operator with possible communication intention spawning
            Then my cycle is done and return control to the main system
    If no operator triggered (for all activated tasks)
    Then my cycle is done and return control to the main system

---

Figure 4.7 Agent execution cycle

For example, in one cycle an agent *a1* can query another agent *a2* for resources for its task in a means-end manner, i.e. the executed operator will spawn a communication intention (to

query *a2* for resources) in order to acquire the resources it needs to proceed with the task advancement. In the next a1's cycle, if we assume that *a2* answered in the meantime, *a1* will run the 'understanding' filters on the received message, complete the open conversation and update its beliefs (e.g. with the values of resources) in the 'Process received messages' phase. When it reaches the 'Operator processing' phase, the operator will be able to advance the task into the next abstract state (e.g. state of having all resources for the task execution – have-all-resources(task-t)).

4.2.3 Task Operators

The task operators are used as the driving force for moving the task through the problem solving space. Each task goes through certain transitions and abstract status states as it is shown in Figure 4.8.

Initially, the task is in the *not-attempted* state, i.e. its execution has not started yet. From the *not-attempted* state a task can move into the *possible* state if it has all the needed resources and they are semantically valid. In the case this is not true, the task will move into the *not-possible* state. Those transitions from the *not-attempted* state are done with the abstract task operator T1 (Table 4.3). The *not-possible* state means that a task ends in the final *failed* state, by means of the T2 abstract task operator (Table 4.3).

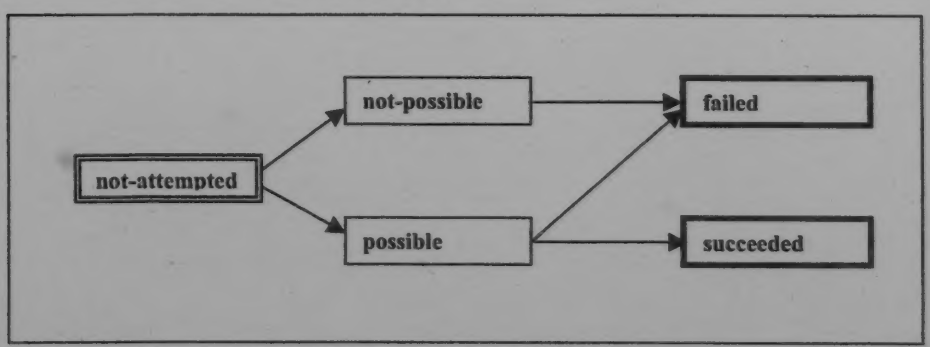


Figure 4.8 Task state transition diagram with abstract task status states

In the case of a task being in the *possible* state, the abstract task operator T3 (Table 4.3) will move it into one of the final states, namely *succeeded* or *failed*. Whether a task succeeds or fails, it depends on the success of the task execution and whether the produced results satisfy the task postconditions or not.

At any moment a task that moves through the abstract status states, due to the relevant events in the task structure, can become *irrelevant*. For example, if a task *t1* is a sibling task (i.e. it shares the same parent task) with a task *t2* and they are connected through the 'and' decomposition, then in case *t1 fails t2* will become irrelevant (in the task structure), since the 'and' decomposition expresses 'all-or-nothing' principle for sub-task execution and relevancy.

Task operator **T1**
Transitions from *not-attempted* to *not-possible* state:

```
if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=not-attempted
        and not-have-all-resources(task-t)
then    status(task-t)->not-possible

if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=not-attempted
        and have-all-resources(task-t)
        and not-valid-all-resources(task-t)
then    status(task-t)->not-possible
```

Transition from *not-attempted* to *possible* state:

```
if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=not-attempted
        and have-all-resources(task-t)
        and valid-all-resources(task-t)
then    status(task-t)->possible
```

Task operator **T2**
Transition from *not-possible* to *failed* state:

```
if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
```

```
        and status(task)=not-possible
then    status(task-t)->failed
```

Task operator **T3**
Transition from *possible* to *succeeded* state:

```
if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and have-all-results(task-t)
        and valid-all-results(task-t)
then    status(task-t)->succeeded
```

Transitions from *possible* to *failed* state:

```
if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and not-have-all-results(task-t)
then    status(task-t)->failed

if      agent(self, task-t)
        and able(self, task)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and have-all-results(task-t)
        and not-valid-all-results(task-t)
then    status(task-t)->failed
```

Table 4.3 Abstract task operators (in pseudo code)

Table 4.3 shows the abstract task operators; they (and the other operators discussed below) consist from two parts, namely the operator trigger and the operator execution body. The trigger is matched against the agents beliefs, and if it *matches* them (i.e. gets triggered), then the operator body is executed and different actions are performed, such as belief updating, spawning a communication intention or domain-specific code execution.

So far we have discussed the abstract task operators that move a task from one status state to another. But, between the status states there are additional internal problem solving states that keep track of reaching the next status state. These are called *supporting task operators*.

The supporting task operators are responsible for the task resources handling, for the task execution (whether it requires decomposition into subtasks or the domain code execution), and for the task post-processing, when a task is in the final state but there are social obligations to be fulfilled, such as sending results to the delegator agent who asked for it. We look into the supporting operator S1 that is responsible for checking the availability of the resources needed for the task execution, and fetching them through the conversation with other agents. The rest of the supporting operators are presented in the Appendix C.

Supporting operator S1
Resource availability and fetching

```
if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=not-attempted
        and have-all-resources(task-t)=null
        and not-have-all-resources(task-t)=null
then
        check for all resources
        if check shows I have values for all resources
        then I have all resources
        else if I don't have some resources and they are not available
        then I don't have all resources
        else if I don't have some resources but they are available
        then intend to query agents who I believe know these resources
```

Table 4.4 Supporting task operator S1 (in pseudo code)

Supporting task operator S1 triggers in case a task is in the initial state, and the availability of the resources is mostly unknown. The function of the S1 operator is to determine whether needed resources are available internally for the agent, if that is not true, then if they are available through the conversation with the agents that should know them according to the models of other agents. If all resources are found then the agent is ready to move the task further into the resource validation and the execution with the other operators. If not, the failure to provide the resources for the task means that a task will fail. This operator is one of the supporting operators that trigger communication plans for generating messages.

4.2.4 Communication Plans

The communication plans deal with the message understanding and the message generation as shown in Figure 4.1. They rely on the structures behind the speech acts that we presented in the previous chapter, and they deal with the *inform*, *disconfirm*, *query-ref* and *not-understood* FIPA ACL message types.

A communication plan gets instantiated and executed when a message is received or when a message is to be generated. First, let us look into the message generation case. A message may be generated for two essentially different reasons. In the first case, it can be motivated by a task operator that needs some information to proceed with the task advancement. For example, the supporting task operator S1 we presented in Table 4.4 has a duty to deal with the resources for a task, i.e. to acquire all the needed resources or to declare them not available. Once it reaches the point when it should acquire a resource from another agent it spawns an intention to query that agent about the resources. In the second case, an agent can generate a so-called proactive *inform* message. For example, if an agent is performing a task but it turns out that it failed, this agent can inform another agents about the status of its task in case those agents need its results or resources. Overall, a communication intention is spawned from the problem solving module, i.e. from the abstract task operators. For each potential communication intention there are checks if such communication intention already exists and if there is a communication pathway towards the intended receiver. After this stage is passed, the communication module checks against agent's belief state if all the preconditions are satisfied. Following this, the message is generated and sent to the receiver agent. Routines for generating particular message types are presented in Appendix D. The agents cannot generate messages that are illegal or non-pragmatic based on their interaction model.

Second, let us consider message understanding and assimilation process. When a message is received it is tested through the filters that determine if the message is legal and pragmatic (as presented in details below), if it is not then a *not-understood* is generated, and potentially a *not-understood* related conversation is spawned (as described in the next chapter). If the received message passes all the communication plan filters, then it results in a belief update of the receiver agent corresponding to the propositional content or it can also result in the immediate answer (in case of a received *query-ref* message).

Belief update is important part of the message assimilation process and conversations in general because it defines the speech act effect on the receiver; after the message is understood, it triggers belief update and inference process based on the speech act type and the content. For

example, if a message is *inform* about the task results, then the receiver agent will assimilate the actual propositional content (the result of task t is r) into its beliefs about tasks, it will also make inferences such as that the sender knows the result, that the task was successfully finished, and that it now knows the result itself. In an equivalent fashion, the *query*, for example, about a resource will produce the belief that the sender desires to know the resources for its task, and that it also intends to perform its task (which is relevant).

Each message, sent or received, is stored in the belief structure supporting the conversations. Besides the actual message elements, this structure also contains the status of the conversation, which can be *open*, *complete* or *time-out*. *Open* conversation status means that the message was sent and the answer is expected; *complete* means that the answer was received (or not expected at all); *time-out* means that answer was not received in the expected time.

Table 4.5 presents the plans for filtering and checking for illegal and non-pragmatic messages. In the implementation, the message understanding code contains the same form of the message checks as presented in the Table 4.5.

Legal *inform/disconfirm* (sender i, receiver j, proposition p):

if	p is public predicate referring to the sender's task
or if	p is external predicate referring to the receiver's task
or if	p is referring to the sender's attitude
then	*inform/disconfirm* message is *legal*

Legal *query-ref* (sender i, receiver j, proposition p):

if	p is external predicate referring to the sender's task
or if	p is public predicate referring to the receiver's task
or if	p is referring to the receiver's attitude
then	*query-ref* message is *legal*

Pragmatic *inform/disconfirm* (sender i, receiver j, proposition p):

if	public-p=status
then	role(i, j) *must be* (delegator or delegatee or enabler or enablee)
if	public-p=result
then	role(i, j) *must be* (delegatee or enabler)
if	external-p=relevancy

then role(i, j) *must be* (delegator)
if external-p=resource
then role(i, j) *must be* (delegator or enabler)
if p is referring to the sender's attitude
then role(i, j) *must be* (delegator or delegatee or enabler or enablee)
if *inform/disconfirm* message abides by one of the above rules
then *inform/disconfirm* message is *pragmatic*

Pragmatic *query-ref* (sender i, receiver j, proposition p):

if public-p=status
then role(i, j) *must be* (delegator or delegatee or enabler or enablee)
if public-p=result
then role(i, j) *must be* (delegator or enablee)
if external-p=relevancy
then role(i, j) *must be* (delegatee)
if external-p=resource
then role(i, j) *must be* (delegatee or enablee)
if p is referring to the receiver's attitude
then role(i, j) *must be* (delegator or delegatee or enabler or enablee)
if *query-ref* abides by one of the above rules
then *query-ref* message is *pragmatic*

Table 4.5 Message filtering and checking plans

From the pragmatic perspective, there is a difference between *inform* and *disconfirm* messages, i.e. *inform* assumes that the receiver does not have any knowledge about the propostional content, while *disconfirm* assumes that the receiver has some knowledge concerning the propostional content. FIPA ACL specifications (2003) acknowledge this difference to be hard to establish by the agent and therefore they suggest the following:

> [it] is not as strong [requirement] as it perhaps appears. The sender is *not* required to establish whether the receiver knows p [message propositional content]. It is only the

case that, in case that the sender already happens to know about the state of the receiver's beliefs, it should not adopt an intention to tell the receiver something it already knows.

In our implementation of the interaction model, we follow the above recommendations from FIPA and we do not explicitly include this distinction as a hard requirement for the agents. Nevertheless, if an agent happens to be aware of the receiver's knowledge of the intended propostional content of a message, then the agent generates the *disconfirm* instead of the *inform* message. For example, we can have such a situation in case an agent *a1* informs an agent *a2* of *p*; if *a2* believes *not p*, then *a2* can decide to send a *disconfirm p* message to a1 (since it is aware of the a1's knowledge of p). We have examples of such conversations being spawned by some cases of *not-understood* messages, as discussed in the next chapter.

When an agent has an intention to send a message, it has some additional pragmatic constraints it follows, such as making sure that there is not already such existing intention or that it actually has a communication pathway for sending a message to a receiver agent.

4.2.5 Task Domain Abilities

The task domain abilities (or task domain specific code in Figure 4.1) refer to the agent's abilities to execute the domain specific code and to check the constraints specified for a task or a method execution. We assume that the agents are capable of that; a task designer is responsible for this at this moment, although it can be envisioned that agents themselves can participate in the planning and task structure design process, and can declare their own capabilities.

In our flight planning type of a task, domain specific code can be responsible, for example, for obtaining the user information for the desired trip or for searching the database of available flights from different airline companies. Our implementation contains the code that simulates these domain specific capabilities. In other words, there are methods that should be getting user's input through the web interface or performing database search in the real world application, but in our system these methods are either simply returning predefined data, or simulate failure and return error signal.

4.3. Summary

In this chapter, we presented the implementation of our multi-agent system and the details of the agent architecture. Our software agents are using an interaction model in order to cooperate in the problem solving episode while working on a joint task.

Firstly, we discussed the implementation details of the task modeling language used for the representation of the declarative task knowledge, and the ontology that defines vocabularies for different levels of communication, such as communication acts, propostional content and propostional content properties. Secondly, we presented the task operators and communication plans that embody the procedural elements of an interaction model plus the implementation specific approaches.

In the next chapter we discuss the behavior of our system, i.e. its problem solving process with the communication at its core. We explore the regular and *not-understood* conversation cases among the agents, along with the problem solving results generated by the system. Overall, these results show the applicability, flexibility and efficiency of the interaction model for regulating agent communication in the context of the agents' cooperative efforts.

Chapter 5 System behavior

In this chapter we present and discuss the behavior of our multi-agent system that was built using the principles of the interaction model. In the previous chapter we presented the agent architecture and the implementation details of the system, namely the task modeling language, the communication related ontologies, the task operators, the communication plans and the domain specific abilities. Here, we explore the resulting cooperative problem solving behavior, i.e. how agents go on with their individual tasks and how they jointly work to advance the problem solving of the whole shared task structure. The key part is to observe the communication patterns, i.e. the conversations between agents, the way conversations are grounded in the problem solving episode, how the agents understand the messages based on their beliefs related to the task properties and other agents' roles. Moreover, we can observe the situations in which an agent does not understand a received message, and it generates the *not-understood* response including the reason for not understanding the message. Depending on the definition of the legal and pragmatic messages in the interaction model, an agent that receives a *not-understood* can answer and engage in the so-called 'not-understood' conversation or it can just update its beliefs (about the sender).

Firstly, we present the general behavior of the system as the introduction into the test cases that follow. Secondly, we present the test cases with description of conversation traces that show that the interaction model supports the system behavior we advocate. Thirdly, we discuss the *not-understood* responses and the *not-understood* conversations, and we present the corresponding conversation traces. Lastly, we summarize and discuss the overall results.

5.1 Overview of the system behavior: Problem solving process and communication

This section discusses how an interaction model translates into the conversation patterns and how it reflects in the agents' beliefs about the problem solving episode and about the roles of other agents. We start with an intuitive discussion of the behavior and communication in our system. The declarative knowledge from the interaction model is used by the agents to initialize their beliefs about the task they will cooperatively work on and their beliefs about other agents.

In order to give a clearer picture of the cooperative problem solving we will follow it through the example execution of the task structure in Figure 5.1. It presents one of our test task structures that is responsible for the air line ticket search; the left ('and') branch retrieves the

origin and destination locations from the user, and the right ('or') branch is responsible for finding the corresponding air lines. (Recall that the domain specific code is separated from the problem solving mechanisms, and the way the interaction model based agents deal with this kind of structure is independent of a particular domain). This task structure and the modifications of it will be used in several examples in this chapter.

Recall a key axiom: an agent will intend its task iff it believes it to be relevant. Once it acquires this belief, then its abstract task operators become triggered, and the agent's problem solving (and communication) behavior starts. When we look at the Figure 5.1 structure, the question naturally arises: How do agents acquire the belief that their particular task is relevant, and how does the overall control of this distributed agent system develop. Put more crudely, how does problem solving *start* in this distributed system?

There are several ways an agent can acquire the belief that its task is relevant (and hence adopt the intention to solve it). First, an agent can come to believe its task is relevant through an external communication action taken by another agent. Assuming the structure given in Figure 5.1, agent ag3 would believe its task T021 is relevant when it receives a query from agent ag1 for the results of its task. Thus, in the simplest way, agents acquire the belief that their task is relevant from a communication action received from a delegator agent (assuming that the communication action is understandable within the definition of the interaction model—an issue we return to later). This would generate a kind of top-down, invocation based behavior of this agent system. The agent at the root node of the task would be initialed with believing that its task is relevant and all other agents would simply wait to be called upon for action on their particular tasks. With the exception of the root node agent, the other agents would be primarily reactive.

Agents might also determine that their task is relevant in a more autonomous, proactive manner, depending on their analysis of the shared task structure. For example, agent ag2 and ag3 could reason that their respective tasks T031 and T032 are relevant to the overall task, because both these tasks are necessary for the completion of T021, which in turn is necessary for the completion of T011. Agent ag2 and ag3 can come to this conclusion if they have access to the entire task structure.

Clearly, there are several issues at play that determine just whether such proactive behavior, based on this kind of reasoning, will actually be feasible. One is the complexity of the task structure and whether an agent can draw any definitive conclusion about the necessity of its task. For example, whether a task is relevant for the task structure or not can depend on some static factors that can be taken into account and some dynamic factors that cannot be (easily) predicted. The static factors are the task structure and the particular decomposition parameters on

the path through the task structure tree, and the dynamic factors such as the failure of tasks and methods. On the other hand, task T034 will become relevant only if task T033 fails (this is based on the simple assumption that 'or' decomposition subtasks execution would be attempted in sequential order). In other words, we can envision that the agents can calculate the probability of their tasks being relevant based on the task structure and the probability of the failure of the other tasks. This is a possible extension to the existing system.

A second issue is whether agents in a large distributed system will in fact *have* a representation of the entire task structure—this is an assumption of our test system here, but it is neither required for the interaction model nor assumed in other frameworks that allow for agents to have merely a "partial view" of the distributed system (e.g. Wagner et al., 1999). Finally, agents in this system could simply be initiated with the belief that their task is relevant, and all agents could simultaneously begin working on their tasks. This can create complex control and scheduling issues, a matter that is not at the heart of the issues we focus on in this work.

For the current system, the agents determine their task relevancy based on the parent task decomposition type, i.e. "if the parent task has 'and' decomposition then my delegated subtask is relevant", "if the parent task has 'or' decomposition then my subtask is not relevant (unless a communication action leads me to infer otherwise)".

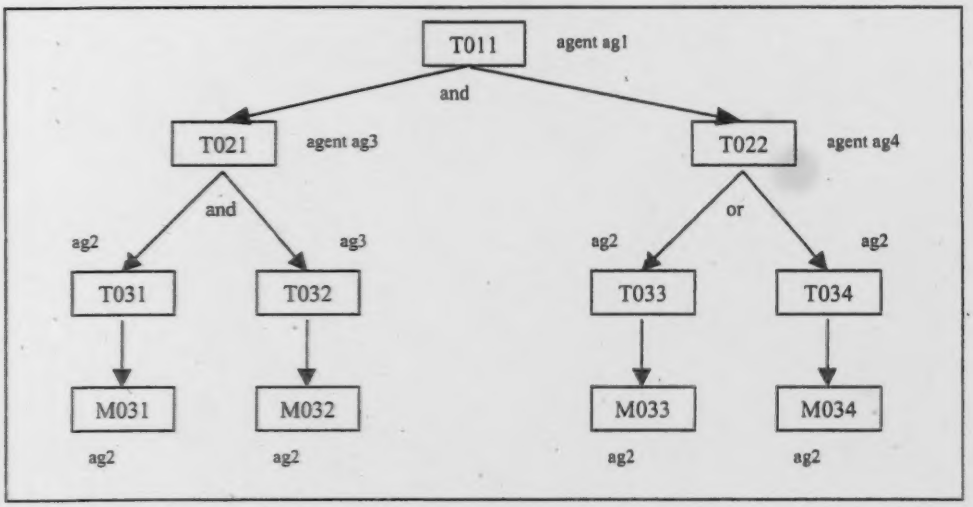


Figure 5.1 A test task structure

Using the Figure 5.1 task structure, agents ag3 and ag2 will, for example, consider tasks T021 and T031, respectively, to be relevant. They will begin problem solving activities for those

tasks even before being requested for results from agent ag1. However, agent ag2 — also assigned to tasks T033 and T034—will consider them each irrelevant at the beginning of the problem solving episode).

As agent ag1 moves the top task through the problem solving states (according to the problem solving operators described in the previous chapter), it will reach the moment when in order to finish its task it needs the results from its subtasks. This is the moment when the agent's communication plans get invoked and the agent sends query message to the agents responsible for the subtasks of its task. The queried agents will (cooperatively) try to answer as soon as possible, that is, when they finish executing their tasks and have the result(s) they have been asked for. It means that the queries for the results will propagate down the task structure tree (from delegator agents to delegatees), and, eventually, the answers will be sent upwards to the querying agents. This will go until all the results for the top task have been received by the 'top' agent, and it will proceed with finding out if the results satisfy the task's postconditons and hence if the task succeeded or failed. This is the simplified view of executing the task structure as a top down invocation, but it can get arbitrarily more complex with agents being proactive about their tasks as noted before. Following our example of Figure 5.1 task structure, agents ag3 and ag2 will start proactively executing tasks T021 and T031, respectively (without being queried for results by ag1). Based on that, the proactive *inform* messages can be sent between agents and they can be understood based on the interaction model beliefs. The examples of such proactive *inform* messages could be, for example, ag3 proactively sending result of T021 to ag1, or ag4 proactively sending resources for T033 to ag2. The actual occurrence of the proactive *inform* messages in the system depends on the timing, i.e. the sender has to be in position to acquire the communication intention to send an *inform* before the receiver sends the *query* message (as it can be seen in the traces we present below).

It is also important to understand, generally, the conditions under which agents will stop problem-solving on their tasks. An agent will stop problem solving if it believes its task is in a final state of succeeded or failed, or if it became irrelevant. In Figure 5.1, agent ag1 will infer that task T022 is irrelevant, if it learns from agent a3 that task T021 has failed.

The overall task structure can succeed even if some of the tasks or methods fail, e.g. if M033 fails, the alternative problem solving path will be pursued and the alternative method M034 will be executed. If it happens that all the alternative paths and methods fail (e.g. if M034 also fails) the whole task will end with its final status as 'failed'. Even in those situations, the agents continue communicating in order to spread the relevant knowledge and they end the problem

solving episode with the consistent beliefs. This is the rough outline of the behavior that will be presented in more details in the next section.

5.2 Test cases of the regular behavior

The main thrust of this research is the way in which an underlying interaction model defines legal and pragmatic (understandable) messages, and hence indirectly defines occasions for not-understood messages. However, we first wish to demonstrate that the interaction model supports "normal" behavior, i.e., that the implemented agent system does solve the overall task or recognizes it cannot be solved.

Table 5.1 defines seven test scenarios that exercise the problem solving operators and communication plans described in the earlier chapters.

Scenario	Task structure	Description
1	Figure 5.2	Agent ag1 initialized with task T011 relevant; top-down invocation; all methods succeed.
2	Figure 5.2	Agent ag1 initialized with task T011 relevant, ag2 initialized with T021 relevant, ag3 initialized with T022 relevant; all methods succeed. Note: agent ag4 is responsible for method M022
3	Figure 5.2	Agent ag1 initialized with task T011 relevant, ag2 initialized with T021 relevant, ag3 initialized with T022 relevant; method M021 fails.
4	Figure 5.3	Agent ag1 initialized with task T011 relevant; top-down invocation; all methods succeed
5	Figure 5.3	Agent ag1 initialized with task T011 relevant; top-down invocation; method M021 fails.
6	Figure 5.1	Agent ag1 initialized with task T011 relevant, ag2 initialized with T031 relevant, ag3 initialized with T021 and T032 relevant, ag4 initialized with T022 relevant; all methods succeed.
7	Figure 5.1	Agent ag1 initialized with task T011 relevant, ag2 initialized with T031 relevant, ag3 initialized with T021 and T032 relevant, ag4 initialized with T022 relevant; method M031 fails.

Table 5.1 Test scenarios

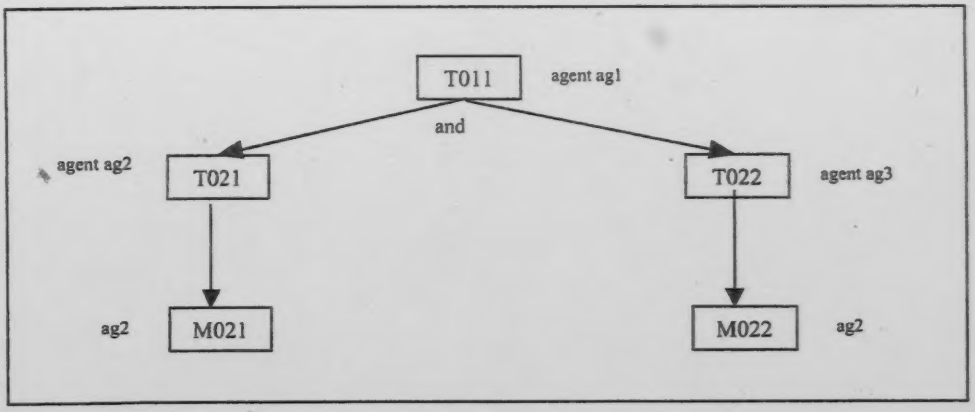


Figure 5.2 A simple test task

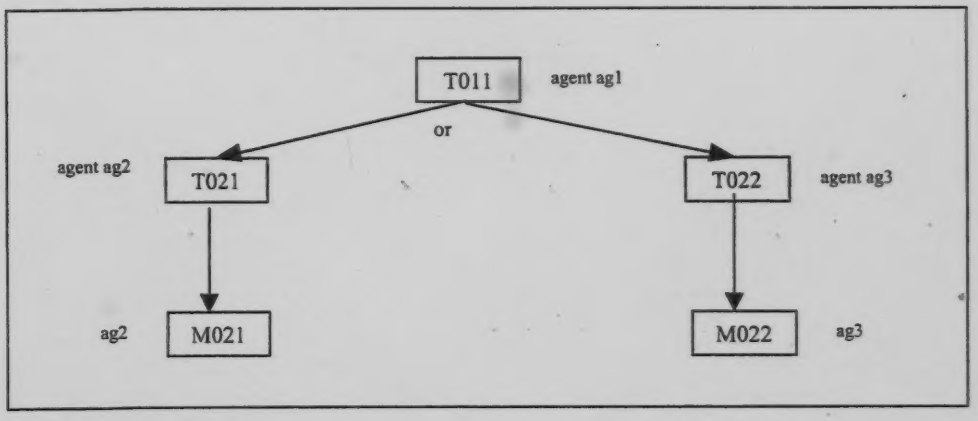


Figure 5.3 A simple test task

Appendix E gives the complete message exchange between all agents, in order, with relevant annotations. Here, we briefly discuss a few features that emerge from these traces for each scenario.

Scenario 1. This scenario presents the simple task structure being executed in the top-down invocation manner. Only top task T011 is initialized as relevant, and its agent ag1 starts the corresponding problem solving activities. The rest of the tasks become relevant when the agents responsible for them receive query for results from their respective delegators. The communication patterns follows the form *query—inform*. All tasks in the task structure were initiated. Eight total messages were exchanged. All tasks and the whole task structure were

finished successfully; all the involved agents have the beliefs about their tasks reflecting this status; for example, ag2's final (relevant) beliefs about its (successful) T021 task are: `[done, T021], [status, succeeded, T021], [have-all-results, ag2, T021].`

Scenario 2: In this scenario, we have two other tasks, T021 and T022, proactively initialized as relevant by their respective agents ag2 and ag3. (It means that ag2 and ag3 started the problem solving activities on these tasks, without being queried for results from their delegator ag1.) The first message in the conversation trace shows ag3 querying ag1 for resources for task T022 (before ag1 queried for T022 results from ag3). Message 6 gives an example of an proactive inform. Agent ag3 proactively informed ag4 about the resources needed for M022 after it asked for M022's results. After the message 5, ag3 inferred that ag4 intends to perform method M022, and that it desires the resources needed for that. All tasks in the task structure were initiated. Nine total messages were exchanged. All tasks and the whole task structure were finished successfully; all the involved agents have the beliefs about their tasks reflecting this status (the same as in previous scenario).

Scenario 3: Interesting behavior in this scenario is produced by the simulated failure of the method M021. Consequently, T021 also fails and ag2 informs ag1 about the T021's failure (message 4). Agent ag1 infers that the other 'and' subtask T022 is irrelevant and sends *disconfirm* to ag3 (message 5) since based on the message 1 it believes ag3 believes T022 to be relevant; after receiving message 5 ag3 stops believing T022 is relevant and stops executing it. Method M022 was never initiated. Five total messages were exchanged. Method M021 and tasks T021 and T011 failed; task T022 became irrelevant and ag3 stopped intending to perform it. This is reflected in the agents beliefs such as ag2's final beliefs about its failed tasks: `[not-have-all-results, ag2, T021], [status, failed, T021], [done, T021], [not-have-all-results, ag2, M021], [status, failed, M021], [done, M021]`. The execution of the task structure failed as reflected in the status of the top task T011 in ag1's final beliefs: `[not-have-all-results, ag1, T011], [done, T011], [status, failed, T011].`

Scenario 4: This scenario shows simple situation of two branches with 'or' decompositions. As shown in the conversation trace, top agent ag1 asks ag2 for results of task T021. After successfully finishing T021 and M021, ag2 informs ag1 about results. Method M022 and task T022 were never initiated. Two total messages were exchanged. Task T011 (and the whole task structure) successfully finished; ag1 has the following final beliefs: `[have-all-results, T011], [valid-all-results, T011], [done, T011].`

Scenario 5: This scenario shows the same simple task structure as in the above case, but the first branch fails because M021 and, consequently, T021, fail. This can be observed in the messages 1 and 2. After being informed that T021 has failed (message 2), ag1 asks ag3 for the results of the alternative sub-task T022 (message 3). Agent ag3 successfully executes T022 and inform ag1 about the result (message 4). Overall, M021 and T021 fail; M022, T022 and T011 are finished successfully. Four total messages were exchanged. Since task T011 was successful, the whole task structure is successful.

Scenario 6: This scenario shows the behavior of the agents working cooperatively on somewhat more complex task structure. The conversations and inferred beliefs still follow the same patterns as above. The interestingness of this scenario is to show that agents are successfully engaged in a number of conversations through the problem solving process, sometimes concurrently. For example, ag2 is working on T031, M031 and M032 (in the first part of the episode), and it has 'open' simultaneously two conversations with ag3 concerning different tasks (messages 6, 7, 8 and 9). Agent ag3 has, besides those conversations with ag2, additional two open conversations with ag1 (messages 4, 5, 10 and 11). All tasks and methods finish successfully, except for the T034 and M034 that were never initiated. Eighteen total messages were exchanged.

Scenario 7: This scenario differs from the previous one in the fact that method M031 fails and the agents' conversations take on different direction. After ag2 informs ag3 that T031 failed (message 9), ag3 infers that T021 fails and informs ag1 about that (message 10). Consequently, ag1 infers that T011 has failed, and that T022 is not relevant any more and it tells that to ag4 using *disconfirm* message type (message 11) because it inferred before that ag4 believes T022 is relevant (from message 1). M031, T031, T021 and T011 have failed; M032 and T032 have finished successfully; T022 has been deemed irrelevant; T033, T034, M033 and M034 were never initiated. Eleven total messages were exchanged.

The above test scenarios show that the agents can handle the communication based on the interaction model principles. The system deals with the protocol-like communication, but agents also understand the messages that present the sender's proactive attitude in sending information that was not asked for. On a *deeper* level, agents ground the understanding of the content (relatively to a particular speech act) in the problem solving context. The test scenarios and conversation traces also show that interaction model deals seamlessly and naturally with the multiple conversations, even if they are interrelated. The conversations can be related in such a way that one conversation spawns another one (with the same or some other agent), e.g. delegator agent asks for the results of the delegatee's task, it can (indirectly) spawn the query from the

delegatee about the results of its subtask towards its own delegatees. Another case is when the content from one conversation is relevant for another conversation, e.g. *ag3* has not yet finished the conversation with its delegator about its task's results, but its delegatee informed it in the mean time that its ('and') subtask has failed, and therefore ag3 has to inform delegator of its task failure.

The code that handles the conversations is written in Java, that is, Java classes that form the communication framework module (CommFrame java package) (Figure 4.1). The communication module contains two main classes, namely the class for message generation (SA.java), and the message understanding and assimilation class (Input.java). In order to handle the conversations successfully, those two classes are heavily relying on the various other Java classes: utility classes within the communication module, and utility classes within the general utility java package (MiscUtil java package), and classes for support and manipulation of agent belief state (Belief java package) (Figure 4.1). Communication classes are also dependent on the abstract problem solving engine (abstract task operators, with the main Java class Operators.java) (Figure 4.1). The working assumption of this research is that messages and conversations are grounded in the problem solving context, therefore for a message to be properly handled, the problem solving engine must have had properly generated the problem solving context, the agent's beliefs about it and about other agents. In order to generate the above scenarios and the scenarios presented in the next section, the multi-agent system has to be run on the appropriate input XML files containing the desired task structure and other interaction model components, and with appropriate task domain specific code.

In this section we have shown the system's 'normal' behavior, in the next section we present the *not-understood* cases.

5.3 *Not-understood* responses and conversations

5.3.1 Overview

In order to present the *not-understood* responses and how they are grounded in the declarative and procedural knowledge based on the agent interaction model, we take a more detailed look into the general message assimilation routine. Such a routine parses an incoming message and determines whether a *not-understood* response is justified, by asking these five questions:

1. Is the predicate in the content proposition a public predicate?

2. Is the predicate in the content proposition a defeasible predicate?
3. Is the illocutionary force applied to this predicate allowed?
4. Does the sender hold the proper role relative to the hearer, in order to apply this illocutionary force to the proposition?
5. Are the feasibility, relevance, and success preconditions (assumed to hold on the speaker's part) consistent with the receiver's interaction model and its current belief state about the sender and the task?

The general form of a *not-understood* message type is *(not-understood :sender j :receiver i :content (m reason))*, where *m* is a just-received message from *i*. The five checks listed above may generate a *not-understood* with a *reason* that takes one of the five possible forms:

(not-understood :sender :receiver

:content (*m* (not (public <predicate *p* of *m*'s content>)) |

(*m* (not (defeasible <predicate *p* of *m's* content >)) |

(*m* (not (externally defeasible <predicate *p* of *m's* content >)) |

(*m* (not (permissible-role sender receiver)) |

(*m* (not α)))

The first three of these cases impose the propositional constraints; for example, the sender may send the message whose content contains a predicate that is not a part of the receiver's public vocabulary. The fourth case points towards the sender-receiver role not being permissible from the receiver's point of view; for example, receiver can believe that the sender does not have the appropriate role 'to know that'. The last case includes a proposition α that will describes the mismatch between the receiver's model and the sender's model of what feasibility and relevance conditions are thought to be satisfied. For example, this case might correspond to "It is false that I desire the result of task-23." This could occur if tasks are dynamically reallocated and agent models about task responsibilities are out of sync.

These cases are handled by the filtering plans that are based on the context-sensitive grammars (presented in Figure 4.4 and Figure 4.5). The filtering plans (Table 4.5) contain checks that have to be satisfied for a message to be allowable and understood, if it is not then the check that is not satisfied triggers generation of a *not-understood* message with the reason (in one of the

above forms) based on the not satisfied constraint. Communication module (Figure 4.1) is responsible for *not-understood* responses and conversations.

At this point, we could consider the matter of generating and structuring *not-understood* messages as done. That is, we could have a system that merely generates *not-understood* messages without further message exchange. This is especially true if we regard the *not-understood* messages as true run-time errors, i.e., errors that would bring an agent system to a halt, or at least stop further interaction between two agents.

However, things become more interesting, and complicated, if we try to allow some further resolution of a *not-understood* message through additional message exchange. The reason included with *not-understood* is supposed to be a 'constructive explanation' of the difficulty with message understanding. It would be *most* constructive if it caused some change to the internal state of the agent who sent the original message *m*, such that the agent would not simply regenerate message *m* all over again. In this sense, we can interpret the reason in a *not-understood* as an opportunity to bring two disagreeing models into alignment: the reason constitutes the content of an *inform*. But this raises the question of whose model is to be taken as true—the sender who generated message *m* or the receiver who asserts it is not understandable? We do not have a general answer for this, but we believe that the underlying interaction model could be used to represent application-specific policies about this.

The following sections examine the above mentioned *not-understood* categories with the corresponding test case. The test cases we generated based on the Test scenario 6 (Table 5.1) with appropriate changes (as indicated in each case). *Inform* and *disconfirm* messages behave the same in *not-understood* related situations based on their structures as presented in Table 3.4 and Table 3.5, and the *query-ref* messages behave as a mirror reflection of *informs* based on Table 3.6.

5.3.2 On public vs. private predicates

One of the basic not-understood cases is when an agent sends message content that cannot be resolved within the shared ontology or more precisely, certain propositions might be exchanged, in principle, but have not impact on agent behavior.

Consider this possible message exchange scenario:

	message 1	(inform *i j* (done (*j* ....)))
	message 2	(not-understood *j i* (<message 1> (not (public have-resource))))
?	message 3	(disconfirm *i j* (not (public have-resource)))

X message 3' (not-understood *i j* (<message 2> (public have-resource))))

In message 2, *j*'s reason is that *j* believes that *have-resource* is not a predicate that can be part of the message content. This is the simplistic way for an agent to indicate that it is not programmed to do anything with a belief that involves this particular predicate. This is a subtle distinction from the idea of a common ontology – it constitutes what may be talked about as message content between agents. Beliefs might be exchanged about all sorts of things—but to classify something as a public predicate is to indicate it can be part of message content, and to indicate it can be part of message content implies that receiving a belief about it would actually impact agent behavior.

Now, there are two possible responses that *i* could make. In message 3, *i* aims to revise *j*'s model about whether the *have-resource* predicate is public. Is message 3 allowed? In this particular case, it would be odd to imagine that message 3 is at all sensible, because it is equivalent to agent i aiming to change agent j's beliefs about what predicates agent j can understand as message content.

The second possible response is message 3', where *i* tells *j* that *j*'s *not-understood* message is not understandable to *i*. Within our framework, this *cannot* happen, because to do it is equivalent to *i* informing *j* what *j* already knows from message 1 (that *i* believes that *have-resource* was a public predicate). Since agent *i* already has an explicit belief about the existence of *j* belief about the intended content proposition, this would violate a feasibility precondition for *inform* (again regarding the reason as the content of an *inform*), and agent *i* would decide to use *disconfirm* instead (as explained at the end of Section 4.2.4).

In order to test this case and the cases of the illegal messages in the next two sections (Sections 5.3.3 and 5.3.4), we needed a way to generate illegal messages that were not necessarily grounded in the implemented problem solving layer (that spawns communication intentions) and that would not pass all the agent's communication module checks for speech act preconditions (as explained in Section 4.2.4). In order to still generate those illegal messages and to observe agent's behavior in those cases, we developed a separate code that would generate and send the illegal messages in the name of some existing agent (the code is written in java class DummyAg.java). Based on this, we produced the following test case:

a) (inform :s ag4 :r ag1 :c have-resource ag4 origin T022 :cid 7001)

b) (not-understood :s ag1 :r ag4

:c ((inform ag4 ag1 (have-resource ag4 origin T022) 7001)
(false (public have-resource)))
:cid 7001)

5.3.3 On defeasible vs. fixed predicates

This case deals with the sender trying to change the receiver's beliefs about the predicate that is fixed, i.e. non-defeasible (the receiver is programmed not to change its beliefs concerning this predicate), and therefore such a message presents an illegal action. The following test case demonstrates such situation:

a) (inform :s ag3 :r ag1 :c agent ag1 T033 :cid 7001)

b) (not-understood :s ag1 :r ag3
:c ((inform ag3 ag1 (agent ag1 T033) 7001)
(false (changeable agent)))
:cid 7001)

This case was generated because ag1 considers predicate *agent* not to be defeasible (changeable) based on its interaction model (Table 3.2). Here, ag3 is attempting to inform ag1 that ag1 it the agent responsible for task T033. Intuitively, this corresponds to the idea that tasks cannot be reassigned dynamically (of course, if this could occur, then the interaction model would permit *agent* to be defeasible). Message *a)* was generated by the illegal message generator (DummyAg.java) as explained in the previous section.

5.3.4 Is the illocutionary force applied to the predicate allowed?

We next turn to the case in which the illocutionary force applied to a proposition is not allowed under the interaction model. One of the intuitive examples is an agent informing another agent what that second agent's task status is. Consider, then, this message exchange.

message 1 (inform *i j* (status (*…j's task* ....)))
message 2 (not-understood *j i* (<message 1> (not (external status))))
? message 3 (disconfirm *i j* (not (external status)))
X message 3' (not-understood *i j* (<message 2> (external status))))

In message 2, *j*'s reason is that *j* believes that *status* is not an external (defeasible) predicate, i.e., one whose truth status can be updated by a communication action from another agent (it makes sense that only an agent responsible for a task knows and can update its status). Hence *i*'s message 1 is not allowed under *j*'s interaction model.

Now, there are two possible responses that *i* could make. In message 3, *i* aims to revise *j*'s model about whether the *status* predicate is external. Is message 3 allowed? It depends on the underlying interaction model: if the predicate *external* is itself an external, defeasible predicate, then message 3 is allowed. Otherwise, it is not. (This does not resolve the more general matter of whose model should be taken as correct, but that may be regulated via agent roles that make sense within the realm of a given application). The second possible response is message 3', where *i* tells *j* that *j*'s *not-understood* message is not understandable to *i*. Within our framework, this cannot happen, because to do it is equivalent to *i* informing *j* what *j* already knows from message 1 (that *i* believes that *status* was an external predicate). This would violate a feasibility precondition for *inform* (again regarding the reason as the content of an *inform*) and would not be generated by the communication plans as explained in Section 5.3.2.

Here is the relevant message exchange:

a) (inform :s ag3 :r ag1 :c status 'failed' T011 :cid 7001)

b) (not-understood :s ag1 :r ag3
:c ((inform ag3 ag1 (status 'failed' T011) 7001)
(false (external status)))
:cid 7001)

There is a second case in which agent communication is outside the scope of what is allowed by the interaction model. Consider the case where one agent asks another "What is the status of my task?" Now, this makes no sense. I cannot ask you what the status of my task is (has it failed? is it succeeded?)—only an agent can update its beliefs about its own tasks status—only it can decide whether its task has failed or succeeded.

We tested this case with the following message exchange:

a) (query-ref :s ag1 :r ag3 :c status T011 :cid 7001)

b) (not-understood :s ag3 :r ag1
:c ((query-ref ag1 ag3 (status T011) 7001)
(false (external status)))
:cid 7001)

In the above case, it is illegal for *ag3* to send an inform with the predicate it is being queried about, because it would mean that it is trying to influence the receiver's beliefs which only the receiver has right to update (according to the interaction model). Since it is not legal for *ag3* to send answer to the received query, it does not understand why *ag1* sent this query in the first place.

5.3.5 Proper agent role to change a defeasible predicate via communication

The next case for violations of an interaction model concerns the relative role of the sender to the receiver, in order to apply a particular illocutionary force to a proposition. Our intuitive example from Table 3.4 was "According to my model of you, you cannot know *p*".

Let us look at the example of the situation where *i* sends proactive *inform* message to agent *j*. According to its model of agent *i*, *j* might not believe that an *inform* feasibility precondition holds for agent *i*, e.g., that agent *i* has reason to know the results of task-23. If only agents assigned to a task can know task results and *j* does not believe *i* is the agent assigned to task-23, then it might generate message 5 in response to message 4:

message 4 (inform *i j* (result-of (task-23 ....)))
message 5 (not-understood *j i* (<message 4> (not (agent (*i* task-23)))
? message 6 (disconfirm *i j* (not (agent (i task-23)))

Message 6 could be a legal continuation of this exchange, *iff* the interaction model defines *agent* to be an external, defeasible predicate and (by the associated axioms) that an agent is the final authority on what its task assignments are (i.e., it can update beliefs held by others about its task assignments). Here is the relevant message exchange:

a) (inform :s ag2 :r ag1 :c result origin 'Toronto' T021 :cid 7001)

b) (not-understood :s ag1 :r ag2

:c ((inform ag2 ag1 (result origin 'Toronto' T021) 7001)
(false (agent ag2 T021)))
:cid 7001)

c) (disconfirm :s ag2 :r ag1 :c false (agent ag2 T021) :cid 7001)

In order to allow this conversation and the last (*disconfirm*) message we changed the interaction model in such a way to make predicate *agent* to be externally defeasible, and in order to define the belief revision we added the axiom in agents reasoning (implicitly in the code) that states that "an agent is the final authority on what its task assignments are". As a result of that and of the above message exchange, ag1 updates its beliefs about ag2 as being responsible for task T021 and accepts the results it received with the proactive *inform* message (message *a*).

5.3.6 Interaction model mismatch between agent roles and dependencies

The next message exchange illustrates a case where agent *i*'s model of task and agent interdependencies is incorrect, and *i* (proactively) informs *j* of some result *j* does not need:

message 4 (inform *i j* (result-of (task-23 ....)))
message 5' (not-understood *j i* (<message 4> (not (desire (*j*, result-of (task-23...))))
X message 6' (disconfirm *i j* (desire (*j*, result-of (task-23)))

According to the interaction model, message 4 is not understood because, from *j*'s viewpoint, the contextual relevance conditions for this *inform* should not be holding for *i*: *j* does not need to know the result of task-23 and *i* apparently believes otherwise. The *not-understood* reason, taken as an *inform* from *j* to *i* about *j*'s beliefs, is legal under the interaction model. This can cause agent *i* to update its model of agent *j*'s responsibilities. But agent *i* cannot attempt to revise agent *j*'s belief state with message 6, because by the model, *i* cannot change *j*'s state about what *j* desires. Here is the relevant test case (where *ag2* falsely believes that *ag3* is its delegator in relation to task T022):

a) (inform :s ag2 :r ag3 :c result airline 'Air Canada' T022 :cid 2005)

b) (not-understood :s ag3 :r ag2
:c ((inform ag2 ag3 (result airline 'Air Canada' T022) 2005)
(false (desire result ag3 T022)))
:cid 2005)

5.4 Summary and Discussion

In this chapter, we presented the behavior of the system that is developed based on the interaction model, as presented in the previous chapter. The results produced by the test cases have shown that agents can ground the message understanding and generation in the underlying interaction model including the dynamic problem solving context, and that communication is successfully used to move the task towards the more advanced problem solving states. The regular system behavior and communication is described as a situation in which agents' beliefs about the interaction model are in alignment, and we assume that there are no dynamic changes in the task structure. If that is not the case, the *not-understood* answers and conversations can be observed. We have discussed the test scenarios that produced both regular behavior and such episodes where agents did not agree on the underlying interaction model, which resulted in *not-understood* messages and sometimes the whole conversations were spawned.

In the presentation of the *not-understood* episodes, we mainly focused on the *inform* messages. In the cases with message exchanges involving *query* many of the same issues arise. In the simple scenarios considered here, an agent may reply with a *refuse* to a *query* concerning α if it cannot resolve, via its underlying interaction model, that the inquiring agent has reason to want to know α. By FIPA definition, the *not-understood* is also applicable answer to the unresolved *query* messages, and we use it in our system since it stresses the misunderstanding between agents (that is based on the underlying interaction models being out of sync) and it, intuitively, offers the possibility of resolving the misunderstanding in the same way as with the *inform* cases presented above. In this way, an interaction model can be used to enforce certain privacy constraints on information exchanged in a multi-agent system.

The *not-understood* message leads to the issue of aligning the mismatched agents' beliefs and it poses the problem of whose beliefs should be updated. Our simple proposal states that agents are responsible (and they are final authorities) for their own beliefs, intentions and desires, and only the so-called externally defeasible predicates can be changed through the communication actions. Our solution makes the *not-understood* message more than just an error

message that signals the belief mismatch and halts the task execution, it enables agents involved in the communication to synchronize their beliefs; the result is that the agents will bring the joint task to a final state, whether the task fails or succeeds. We believe that the success performance can be enhanced more if the agent interaction patterns are extended with the re-planning abilities, but this is the matter that belongs in the next chapter among the future directions.

In general, we presented simple scenarios showing how agents can use *not-understood* as means for aligning disparate belief states. Belief states of cooperating agents could diverge during problem solving, through lost messages or possibly through task reallocation during run time. So it is not unreasonable to consider that this simple belief revision—triggered through communication exchanges—might be necessary for agents to adjust to changes in their task environment. We included the proposed belief revision principle in our implementation. Overall, the interaction model implementation gives promising results considering the task based communication and cooperative agent interaction.

Chapter 6 Discussion and Future Directions

6.1 Conclusion and Discussion

The contributions of this thesis can be summarized as follows. We have taken seriously the notion that communication actions are attempted acts upon the internal state of the receiver. We used the success, feasibility, and relevance conditions— which impart semantics to agent message types — to propose an interaction model for normative agent communication. We expanded this interaction model to include notions of agent role, agent authority, and agent interdependencies—notions that are important in modeling large scale multi-agent systems. Having defined this interaction model, this work then defined deviations from normative communications, i.e. classes of error conditions that are signaled by communication actions that are not legal according to this interaction model. We were then positioned to structure the cases under which an agent can send a *not-understood* message, coupled with a reason that corresponds to a particular violation. As such, the *not-understood* message constitutes a kind of run-time error that could result when agents' models of each other and their cooperation are not synchronized. These ideas were explored in a small scale multi-agent system, in which all agents accessed a declarative XML representation of the interaction model. The thesis further explored the possibility that belief update of agent models may take place during run-time, as a consequence of receiving a *not-understood* message. The interaction model approach can also serve to constrain the kind of belief updates that are allowed, should a run-time correction to models be allowed.

To employ this sort of approach, a designer must make a commitment to an abstract language for describing that portion of internal agent states which constitute the world which a communication act is attempting to alter. This language structures the :content field portion of an agent message. It is also used to describe the illegality of a communication action, when a *not-understood* message is generated. In doing so, the designer makes a commitment to employing an 'agent paradigm' insofar as s/he commits to representing some portion of the internal agent state as its perception of its current state, a state which includes its interdependencies on other agents which have particular properties. This approach assumes that peer-to-peer communication by possibly autonomous and proactive problem solvers is an important design element to multi-agent systems. The approach also assumes that the interaction model is shared by all agents in the system.

Our working assumption is that understanding and hence not-understanding can only be resolved through appeal to *some* sort of model, which includes as its cornerstone a shared task

specification that describes task and agent interdependencies. Essentially, we have been focusing on coordinating or cooperating agents to which an agent designer can impart (through the interaction model) a "deep model" of their joint work and expectations of each other. This is easy in a closed agent system, in which agents are homogeneous, stable, and the agent paradigm is being employed in a distributed problem solving sense. But in such a system, one can argue that there is no real need to adhere to a high level ACL in the first place, for such agents can be programmed to communicate with each other in whatever way the system designer considers appropriate. A good part of the motivation for a high level standardized ACL was for communication among heterogeneous agents, who most likely have *shallow* models of each other. And that is where we come up against the matter of just what such agents will 'talk about'. There is considerable work on specifying ontologies that can support certain web-motivated types of interactions (e.g., service brokering Nodine et al. (2000)), but it still seems to us that such agents must commit to some kind of underlying interaction model. Fixed conversation protocols can certainly be regarded as a simple interaction model, but —as it is generally recognized — such protocols do not constrain message content, only sequences of message types. Some of the interaction model components we have advocated here (e.g., restrictions on whether particular beliefs can be updated or revised through communication actions taken by agents in particular roles) can add a level of semantic check, even without a deeper shared problem solving model.

In either open or closed agent systems, we think that a case can be made for explicit interaction models of the kind we have considered here. It might seem that providing such models is too much overhead. However, there has been increasing interest in extending and applying software engineering methodologies to the agent paradigm (e.g. Wooldridge et al. (2000), Jennings (2000), Odell et al. (2000a)) and we believe that many of the interaction model components can be leveraged through the incorporation of the software engineering techniques into the agent design and development (as described in the last section).

We have employed the interaction model in a multi-agent solution to a simple domain task (airline travel planning through different web sites) as our test application. Our system framework instantiates and deploys agents that follow the interaction model as they interact to execute this task. We test the framework's ability to generate and understand a subset of FIPA ACL messages, to generate and assimilate the *not-understood* messages by perturbing the individual agent models and generating messages to other agents that are illegal or non-pragmatic, from their viewpoints. We have shown that agents can successfully use an interaction model as basis for message generation and understanding, and, more generally, that they can advance the problem solving process through communication based on the interaction model.

Interesting issues emerge when we move beyond the error ontology *per se* and consider *not-understood* messages themselves as opportunities for belief revision during problem solving. Agent roles are crucial to our analysis, because they strongly influence the legal and pragmatically expected belief revision and updates that can take place through *informs* and *disconfirms*. If agent task assignment (and roles) are permitted to change during a problem solving episode, there must be consistent and reliable means for synchronizing agents' respective models of each other in the task context. This is an important consideration, since it relaxes the assumption that agents systems are stable and task allocation does not change during coordinated or cooperative problem solving. We tackled this issue and we used using the *not-understood* conversations to synchronize mismatching agent models based on the simple assumption that agents are final authorities on their own responsibilities and roles. In our implementation, this mechanism allows agents to align their models of each other, but we believe that this issue should be explored more as we propose in the next section.

6.2 Future directions: Agents and Software Engineering

This section discusses the connection between the agent based systems and the software engineering. We are exploring how software engineering techniques can be applied to the agent system design and analysis, and how the agent based system design can improve existing software engineering technologies. This is important because the agents are being proposed as the solution to the problems introduced by the complexity of the modern software engineering design (Wooldridge, 1998).

We believe that the future direction for our research should include more extensive incorporation of the software engineering techniques. The question is whether the agent systems are ready to become the mainstream software engineering paradigm.

Jennings (2000) takes an analytical approach to this issue. He analyzes the compatibility points between the agent based design and the software engineering design. The first point is that the agent-based design implies decomposition of a system into autonomous entities that interact in a flexible and high level manner. There are some advantages of this approach. Firstly, a decomposition is generally a natural way to represent a complex system. This decentralization reduces system's control complexity "since the system's control know-how is taken from a centralized repository and localized inside each individual problem solving component" (Jennings, 2000). Secondly, the agents are able to make the decisions based on their local perception of the situation (recall the TAEMS subjective level that serves as agent's local view of

the task environment), which can have advantages over having a system's control making decisions based on the outsider's perception of the situation. Furthermore, the problem of the unpredictable sequence of events is present when designing the complex software systems. A strong example of this are interactions since "interactions will occur at unpredictable times, for unpredictable reasons, between unpredictable components" (Jennings, 2000). Therefore, the agents with their ability to establish communication in the manner that was not necessarily foreseen by the designer, responding to the run-time circumstances, seem again as a natural solution to the above mentioned design problem of the unpredictable circumstances. This is also another reminder that agents need communication mechanisms that are more advanced then conversation protocols. Also, the problem of centralized coordination is lessened since agents are taking care of it in the run-time manner.

On the level of software modeling, an agent-oriented abstraction highly corresponds to a common abstraction of the complex systems as a collection of the entities that successfully interact and jointly work on the different levels of abstraction.

The disadvantage of the agent-oriented approach is that interactions are unpredictable, and the run-time system behavior becomes hard to predict due to the possibility of the (unexpected) emergent behavior. In order to tackle these issues, Jennings has proposed the *social level* for agents that would be used as a framework for studying the multi-agent systems behavior and enhancing the predictability of them. In summary, Jennings is professing that the agent systems will enter the mainstream of the software engineering because of all the arguments mentioned above.

From the above analysis we can conclude that the agent based software design has a lot to offer to the modern software engineering. In order to have software agents as the actual next paradigm used in the software design, the agent technology has to be connected to the presently used technologies and standards. Which brings us to the next question of the applicability and usefulness of the software engineering techniques and approaches utilized as components of the design and implementation of the agent systems. We will look into this area in the following text.

Gaia is the software engineering based methodology for the analysis and design of the software agent systems (Wooldridge et al. 2000). It stresses the essential part of the (agent) *roles* in the system organization, the importance of the relationships among the roles, and the systematic patterns of the interactions among different roles. Responsibilities are the key attribute associated with a role, and they specify its functionality. A role is also associated with a set of protocols that determine how it can interact with other roles. Gaia methodology supports the *interaction model* concept as a set of protocol definitions that capture the interaction patterns

between roles with the certain dependencies and relationships between them. Overall, Gaia offers useful abstraction concepts for the agent system design.

A number of researchers have taken a direction of relating agent oriented design with the object oriented design (Bryson et al. 1998; Odell et al. 2000a; Flake et al. 2001). Odell et al. (2000a) look at the agents as an extension of the active objects. They developed Agent UML (AUML) for the agent interaction protocol design as the extension of the Unified Modeling Language (UML). Flake et al. (2001) propose another methodology of the multi-agent system analysis and design based on the UML modeling.

Grosof and Labrou (1999) pursue the research direction of representing FIPA ACL messages and a content language in the XML (eXtensible Markup Language) format with the motivation to integrate agent technologies with the Internet driven software techniques.

This section has shown a significant amount of varied research initiatives to relate the agent based systems with the software engineering methodologies. Still, we believe this is just the beginning of the (necessary) integration of the agent and software engineering technologies. We believe that using established technologies and standards (such as XML, UML, etc.) will help agent-based design in focusing more on the issues introduced by the emerging agent technology, while relying on the foundations already proved in the real-world systems.

In this research, we used the XML representation of the agent interaction model's declarative knowledge (task models and communication related ontologies), and our task modeling was inspired by the software engineering approach (Situ and Stroulia, 2000). We see the future of this research in exploring the possibilities of tighter integration with the software engineering methodology.

One future direction is to explore the possibilities of translating the existing procedural knowledge of the agent interaction model (such as the axioms) into the declarative XML representation, possibly using DAML language (DAML, 2003), and to add the XML representation of FIPA ACL messages (Grosof and Labrou, 1999). A step further would be to exploit this declarative and implementation independent representation of the interaction model by utilizing it by the group of agents with the heterogeneous architectures. This is an important future research since we designed the agent interaction model with the idea of agents in the open systems using it.

Another future direction can focus on exploring the ways to re-design the whole agent interaction model in UML or AUML (Odell et al., 2000a), which would allow us to explicitly model the interaction patterns and to explore them as the high-level abstractions. This approach may lead us into the research of the so-called agent social level, where particular agent details are

abstracted away, and the model contains only the essence of the agent social interactions based on the defined interaction model. There is an open question of the connection between the communication and interaction complexity, and the complexity of the agent reasoning system and mental state in general. This research level could be supported by the UML modeling of the interaction patterns between the agents with the different reasoning system and different mental states.

Additionally, this research can be extended by the introduction of the agent planning and run-time re-planning, which would require the appropriate adjustments of the used interaction model and the belief revision mechanism. For example, the planning process requires some way of making the choices and decisions (e.g. choosing an agent for the task between agents with the same or similar capabilities related to that task). It means that a new agent role should be introduced with the responsibility to make those decisions. This implies that an agent is not necessarily the final authority on its responsibilities (as we assumed) since the agent with the 'planner' or 'decision maker' role may have the final word on this in some situations. This requires the additional reconsideration of the belief revision mechanisms; for example, it needs to be specified in which circumstances an agent is the final authority for its responsibilities and in which circumstances it is an agent in some particular role.

Bibliography

Austin, J. L.: How to Do Things with Words. Harvard University Press (1962)

Barbuceanu, M., Fox, M. S.: COOL: A Language for Describing Coordination in Multi Agent Systems. In First International Conference on Multi-Agent Systems – ICMAS, The MIT Press, California (1995) 17-24

Barbuceanu, M., Fox, M.S.: Capturing and modeling coordination knowledge for multiagent systems. International Journal of Cooperative Information Systems, 5(2-3) (1996) 275-314

Barbuceanu, M., Fox, M. S.: Integrating Communicative Action, Conversations and Decision Theory to Coordinate Agents. Proceedings of the First International Conference on Autonomous Agents, Agents 97, Marina Del Rey, ACM (1997)

Bradshaw, J. M., Dutfield, S., Benoit, P., Woolley, J. D.: KAoS: Toward an Industrial- Strength Open Distributed Agent Architecture. J.M. Bradshaw (ed.), Software Agents, AAAI/ MIT Press (1997)

Bratman, M. E.: What Is Intention? In Cohen, P. R., Morgan, J. L., and Pollack, M. E. (eds.). Intentions in Communication, The MIT Press: Cambridge, MA (1990) 15-32

Bratman, M. E., Israel, D. J., Pollack, M. E.: Plans and resource-bounded practical reasoning. Computational Intelligence, 4 (1988) 349-355

Breiter, P., Sadek, M.: A Rational Agent as the Kernel of a Cooperative Spoken Dialogue System. In: Intelligent Agents III (LNAI Vol. 1193). Springer-Verlag (1997) 189-204

Bryson, J., McGonigle, B.: Agent architecture as object oriented design. In Singh, M. P., Rao, A. S., and Wooldridge, M. J. (eds.), The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97), Springer-Verlag (1998) 15-30

Cohen, P.R., Levesque, H.J.: Intention Is Choice with Commitment. Artificial Intelligence 42(2-3) (1990) 213~261

Cohen, P.R., Levesque, H.J.: Teamwork. Nous 35. (1991)

Cohen, P. R., Perrault, C. R.: Elements of a plan-based theory of speech acts. Cognitive Science 3 (1979) 177-212

DAML Agent Markup Language, www.daml.org (2003)

Decker, K.: Task environment centered simulation. In M. Prietula, K. Carley, and L. Gasser (eds) Simulating Organizations: Computational Models of Institutions and Groups. AAAI Press/MIT Press, (1996)

Decker, K., Lesser, V.: Quantitative modeling of complex computational task environments. In Proceedings of AAAI-93, Washington, D.C. (1993) 217-224

Decker, K., Lesser, V.: Designing a Family of Coordination Algorithms. In: Proc. 5th Intl. Conference on Multi-agent Systems. MIT Press (1995) 73-80

Decker, K., Garvey, A., Lesser, V., Prasad M. V. N.: Exploring organizational designs with TAEMS: A case study of distributed data processing. In Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96), (1996) 283-290

Elio, R., Haddadi, A.: On abstract task models and conversation policies. In M. Greaves & J. Bradshaw (eds.) Working Notes of the Workshop on Specifying and Implementing Conversation Policies, (1999) 89-98

Elio, R., Haddadi, A.: On abstract models and conversation protocols. In F. Dignum and M. Greaves (eds.). Issues in Agent Communication. (LNAI 1916). Springer-Verlag (2000) 301-313

Elio, R., Haddadi, A., Singh, A.: Task models for agent conversation policies. Proceedings of Autonomous Agents-2000 (2000) 229-230 [a]

Elio, R., Haddadi, A., Singh, A.: Task models, intentions, and agent communication. Lecture Notes in Artificial Intelligence 1886: Proceedings of the Pacific Rim Conference on AI (PRICAI-2000), Springer Verlag: Berlin (2000) 394-403 [b]

Ferguson, G., Allen, J., Miller, B.: Trains-95: Towards a mixed-initiative planning assistant. In Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96) (1996)

FIPA Agent Communicative Act Library Specification, available at http://www.fipa.org/repository/index.html (2003)

Flake, S., Geiger, C., Kuester, J.: Towards UML-based Analysis and Design of Multi-Agent Systems. International NAISO Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ENAIS'2001), Dubai (2001)

Georgeff, M., Pell, B., Pollack, M. Tambe, M., Wooldridge, M.: The Belief-Desire-Intention Model of Agency. In J. P. Muller, M. Singh, and A. Rao, (eds.) *Intelligent Agents V* Springer-Verlag Lecture Notes in AI Volume 1365 (1999)

Greaves, M., Holmback, H., Bradshaw, J.: What is a conversation policy? In F. Dignum and M. Greaves (eds.), Issues in Agent Communication, LNAI 1916. (2000)

Grosof, B., Labrou, Y.: An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. IJCAI-99 Workshop on Agent Communication Languages, Stockholm, Germany (1999)

Grosz, B., Sidner, C.: Attention, Intentions and the Structure of Discourse. Computational Linguistics, 12(3) (1986) 175 - 204

Grosz, B., Sidner, C.: Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack (eds.), Intentions in Communication, MIT Press, Cambridge, MA (1990) 417 - 444

Grosz, B. J., Kraus, S.: Collaborative plans for complex group action. Artificial Intelligence 86(2) (1996) 269 - 357

Haller, S., Fossum, T.: Using protocols to model mixed initiative interaction. AAAI Workshop on Mixed Initiative Intelligence (1999)

Ingrand, F. F., Georgeff, M. P., Rao A. S.: An architecture for real-time reasoning and system control. IEEE Expert, 7(6) (1992)

Jennings, N.: Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions. Artificial Intelligence (75) (1995) 195 - 240

Jennings, N. R.: On agent-based software engineering. Artificial Intelligence 117 (2000) 277--296

Labrou, Y., Finin, T.: A Semantics Approach for KQML -- A General Purpose Communication Language for Software Agents. Proc. Third International Conference on Information and Knowledge Management (CIKM '94) (1994).

Labrou, Y., Finin, T.: Semantics and conversations for an agent communication language. Readings in Agents, M. Huhns and M. Singh (eds.), Morgan Kaufmann (1998) 235 -- 242

Labrou, Y., Finin, T., Peng, Y.: The current landscape of Agent Communication Languages. Intelligent Systems, 14(2): IEEE Computer Society (1999)

Lambert, L., Carberry, S.: A Tripartite Plan-Based Model of Dialogue. In Proceedings of the 29th Annual Meeting of the ACL, Berkeley, CA (1991) 47-54

Lesser, V. R.: Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture. Journal of Autonomous Agents and Multi-Agent Systems, 1(1) (1998) 89--111

Lesser, V., Atighetchi, M., Benyo, B., Horling, B., Raja, A., Vincent, R., Wagner, T., Xuan, P., Zhang, S.: A multi-agent system for intelligent environment control. In Proceedings of the Third International Conference on Autonomous Agents, ACM Press (1999)

Litman, D., Allen, J.: Discourse processing and commonsense plans. In P. R. Cohen, J. Morgan, and M. E. Pollack (eds.) Intentions in Communication, MIT Press, Cambridge, Ma. (1990)

Lochbaum, K.: The use of knowledge preconditions in language processing. In IJCAI'95, Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Montreal Canada. cmp-lg/9508011. (1995) 1260--1266

Lochbaum, K., Grosz, B., Sidner, C.: Models of Plans to Support Communication. In: Proc. 8th Nat. Conf. on Artificial Intelligence. AAAI Press (1990) 485-490

Nodine. M., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., Unruh, A.: Active Information Gathering in InfoSleuth. Intl. Journal of Cooperative Information Systems 9 (2000) 3-28

Nodine, M. H., Unruh, A.: Facilitating Open Communication in Agent Systems: the InfoSleuth Infrastructure. In Intelligent Agents IV: Agent Theories, Architectures, and Languages, Munindar Singh, Anand Rao and Michael Wooldridge (eds.) Springer-Verlag (1998) 281-296

Nodine, M. H., Unruh, A.: Constructing robust conversation policies in dynamic agent communities. In Issues in Agent Communication, Frank Dignum and Mark Greaves (eds.) Springer-Verlag, (Lecture Notes in AI, v. 1916) (2000) 205-219

Odell, J., Parunak, H., Bauer, B.: Extending UML for agents. In the Proceedings of the Agent-Oriented Information System Workshop at the 17 the National Conference on Artificial Intelligence (2000) [a]

Odell, J., Parunak, H., Bauer, B.: Representing agent interaction protocols in UML. In Proceedings of the Fourth International Conference on Autonomous Agents (2000) [b]

Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T. R., Neches, R.: The DARPA Knowledge Sharing Effort: Progress Report. In Readings in Agents, M. Huhns and M. Singh (eds) Morgan Kaufmann Publishers (1997)

Pitt, J., Mamdani, A.: Communication protocols in multi-agent systems. In F. Dignum and M. Greaves (eds.). Issues in Agent Communication. (LNAI 1916). Springer-Verlag (2000) 160 - 177

Rao, A., Georgeff, M.: BDI agents: From theory to practice. In Proceedings of the First International Conference on Multi-Agent Systems, Menlo Park, California, AAAI Press (1995) 312-319

Rich, C., Sidner, C. L.: COLLAGEN: When Agents Collaborate with People. In N. Huhns and M. P. Singh (eds), Readings in Agents, San Francisco, Morgan Kaufmann (1997) 117-124

Rich, C., Sidner, C. L.: COLLAGEN: A collaboration manager for software interface agents. User Modeling and User Adapted Interaction, 8(3-4) (1998) 315-350

Sadek, M. D.: A Study in the Logic of Intention. In Proc. 3rd Conf. on Principles of Knowledge Representation and Reasoning. Morgan Kaufmann (1992) 462-473

Searle, J. R.: What is a Speech Act. In Black, M. (ed.): Philosophy in America. Cornell Univ Press (1965) 221-239

Searle, J. R.: Speech Acts: An Essay in the Philosophy of Language. Cambridge University Press (1969)

Shoham, Y.: Agent Oriented Programming. Artificial Intelligence (1993) 51-92

Situ, Q., Stroulia, E.: Task-Structure Based Mediation: The Travel-Planning Assistant Example. In the Proceedings of the 13th Canadian Conference on Artificial Intelligence, Montréal, Québec, Canada, Vol.1822 Lecture Notes in Computer Science, Springer Verlag (2000) 400-410

Smith, I., Cohen, P.: Toward a semantics for an agent communications language based on speech-acts, Proceedings of the Annual Meeting of the American Association for Artificial Intelligence (1996)

Smith, I., Cohen, P., Bradshaw, J., Greaves, M., Holmback, H.: Designing Conversation Policies using Joint Intention Theory. Proceedings of the Third International Conference on Multi Agent Systems (ICMAS-98), Paris, France, IEEE Press (1998) 269-276

Sycara, K., Paolucci, M., Shehory, O.: Interleaving Planning and Execution in a Multiagent Team Planning Environment. Tech. report CMU-RI-TR-00-01, Robotics Institute, Carnegie Mellon University (2000)

Sycara, K, Paolucci, M., Kalp, D., Pannu, A., Shehory, O.: A planning component for Retsina agents. Wooldridge, M., Lesperance, Y., Lecture Notes in Artitcial Intelligence, Intelligent Agents VI, Springer (2000)

Sycara, K., Paolucci, M., van Velsen, M., Giampapa, J.: The RETSINA MAS Infrastructure. Journal of Autonomous Agents and Multi-agent Systems (JAAMS) 7 (2001)

Traum, D. R.: Conversational agency: The TRAINS-93 dialogue manager. In Susann LuperFoy, Anton Nijhholt, and Gert Veldhuijzen van Zanten (eds), Proceedings of Twente Workshop on Language Technology, TWLT-II (1996)

Wagner, T., Benyo, B., Lesser, V., Xuan, P.: Investigating interactions between agent conversations and agent control components. In Working Notes of the Workshop on Specifying and Implementing Conversation Policies, Seattle, Washington (1999) 79-88

Winograd, T., Flores, F.: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publishing Corp., New Jersey (1986)

Wooldridge, M.: Agents and software engineering. AI*IA Notizie (Associazione Italiana per l'Intelligenza Artificiale), Vol. 11(3) (1998) 31-37

Wooldridge, M. and Jennings, N. R.: Intelligent agents: Theory and practice. Knowledge Engineering Review, 10(2) (1995) 115-152

Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (3) (2000) 285-312

Appendix A: FIPA ACL semantics

This appendix presents the formal FIPA ACL semantics. In order to inspect FIPA ACL's semantics let us present the necessary components of the Semantic Language used to specify it (FIPA 2003):

The mental model of an agent is based on the representation of three primitive attitudes: belief, uncertainty and choice (or, to some extent, goal). They are respectively formalized by the modal operators B, U, and C. Formulas using these operators can be read as:

- $B_i p$ [agent] *i* implicitly believes (that) *p*,
- $U_i p$ [agent] *i* is uncertain about *p* but thinks that *p* is more likely than $\neg p$ and,
- $C_i p$ [agent] *i* desires that *p* currently holds.

The following abbreviations and action expressions are also important for understanding of the semantics of the performatives (FIPA 2003):

- Done (*a*, *p*) means that *a* has just taken place and *p* was true just before that
- Agent (*i*, *a*) means that *i* denotes the only agent that ever performs (in the past, present or future) the actions which appear in action expression *a*
- Done(*a*) ≡ Done(*a*, True)
- $Bif_i\Phi \equiv B_i\Phi$ OR $B_i\neg\Phi$, i.e. it means that agent either believes Φ or $\neg\Phi$
- $Bref_i\, \iota x\, \delta(x) \equiv (\exists y)\, B_i\, (\iota x\, \delta(x)=y)$
- $Uif_i\Phi \equiv U_i\Phi$ OR $U_i\neg\Phi$
- $Uref_i\, \iota x\, \delta(x) \equiv (\exists y)\, U_i\, (\iota x\, \delta(x)=y)$

Here are the formal semantics of some of the FIPA ACL performatives:

<i, disconfirm (j, Φ)>
FP: $B_i\neg\Phi$ AND $B_i(U_j\Phi$ OR $B_j\Phi)$
RE: $B_j\neg\Phi$

<i, inform (j, Φ)>
FP: $B_i\Phi$ AND $\neg B_i(Uif_j\Phi$ OR $Bif_j\Phi)$
RE: $B_j\Phi$

<i, query-if (j, Φ)> ≡ <i, request (j, <j, inform-if (i, Φ)>)>
FP: $\neg Bif_i\Phi$ AND $\neg Uif_i\Phi$ AND $\neg B_i\, I_j$ Done(<j, inform-if (i, Φ)>)
RE: Done(<j, inform-if (i, Φ)>|<j, inform(i, ¬Φ)>)

The querying speech act is specified as a request for performing an inform speech act. With 'request', sender is requesting from the receiver to perform an action *a*. The action *a* can be any action that receiver can perform.

<i, not-understood (j, a, Φ)> ≡ <i, inform(j, α)>
FP: $B_i\alpha$ AND $\neg B_i$ ($Bif_j\alpha$ OR $Uif_j\alpha$)
RE: $B_j\alpha$
α = Φ AND $(\exists x)\, B_i$ ((ιe Done(e) AND Agent (e, j) AND B_j(Done (e) AND Agent (e, j) AND (a=e))) = x)
α is showing that an event e happened and the that recipient of the not-understood message is responsible for that event.

Appendix B: Document Type Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Decomposition (property+)>
<!ELEMENT ExplicitCommunicationObjects (predicate, modality, role,
performative)>
<!ELEMENT ExternalData (TaskStructure, ExplicitCommunicationObjects)>
<!ELEMENT Method (property+, Precondition*, Postcondition*)+>
<!ELEMENT Postcondition (property+)>
<!ELEMENT Precondition (property+)>
<!ELEMENT Task (property+, Decomposition, Precondition*,
Postcondition*)>
<!ELEMENT TaskStructure (Task+, Method+, Variable+)+>
<!ELEMENT Variable (property+)>
<!ELEMENT able (private, internal, changeable)>
<!ELEMENT agent (public, internal, changeable)>
<!ELEMENT changeable (public?, internal?, fixed?)>
<!ELEMENT child EMPTY>
<!ELEMENT desire (public?, internal?, changeable?)>
<!ELEMENT disconfirm EMPTY>
<!ELEMENT done (private, internal, changeable)>
<!ELEMENT enablee EMPTY>
<!ELEMENT enabler EMPTY>
<!ELEMENT external (public?, internal?, fixed?)>
<!ELEMENT fixed (public?, internal?, fixed?)>
<!ELEMENT have-all-resources (private, internal, changeable)>
<!ELEMENT have-all-results (private, internal, changeable)>
<!ELEMENT have-resource (private, internal, changeable)>
<!ELEMENT have-result (private, internal, changeable)>
<!ELEMENT inform EMPTY>
<!ELEMENT intend (public?, internal?, changeable?)>
<!ELEMENT internal (public?, internal?, fixed?)>
<!ELEMENT know (public?, internal?, changeable?)>
<!ELEMENT modality (intend, know, desire)>
<!ELEMENT not-understood EMPTY>
<!ELEMENT parent EMPTY>
<!ELEMENT performative (query_ref, inform, disconfirm, not-understood)>
<!ELEMENT predicate (relevancy, status, agent, know, desire, resource,
result, intend, able, done, external, internal, public, private, fixed,
changeable, have-resource, have-all-resources, valid-all-resources,
have-result, have-all-results, valid-all-results)>
<!ELEMENT private (public?, internal?, fixed?)>
<!ELEMENT property EMPTY>
<!ATTLIST property
   id CDATA #REQUIRED
   name CDATA #IMPLIED
   component_of CDATA #IMPLIED
   children_list CDATA #IMPLIED
   agent CDATA # REQUIRED
   child_type CDATA #IMPLIED
   cardinality CDATA #IMPLIED
   choice CDATA #IMPLIED
   expression CDATA #IMPLIED
   parent_list CDATA #IMPLIED
   executable_code_path CDATA #IMPLIED
```

```
  data_type (date | object | string) #IMPLIED
>
<!ELEMENT public (public?, internal?, fixed?)>
<!ELEMENT query_ref EMPTY>
<!ELEMENT relevancy (public, external, changeable)>
<!ELEMENT resource (public, external, changeable)>
<!ELEMENT result (public, internal, changeable)>
<!ELEMENT role (parent, child, enabler, enablee)>
<!ELEMENT status (public, internal, changeable)>
<!ELEMENT valid-all-resources (private, internal, changeable)>
<!ELEMENT valid-all-results (private, internal, changeable)>
```

Appendix C: Supporting Task Operators

Supporting operators that may spawn communication intention

Supporting operator S1
Resource availability and fetching

```
if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=not-attempted
        and have-all-resources(task-t)=null
        and not-have-all-resources(task-t)=null
        and there is no such resource-id that not-have-resource(resource-id, task-t)
then
        check for all resources
        if check shows I have values for all resources
        then I have all resources
        else if I don't have some resources and they are not available
        then I don't have all resources
        else if I don't have some resources but they are available
        then intend to query agents who I believe know these resources
```

Supporting operator S3
Task/method execution: decomposition into subtasks or execution of the code

```
if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and have-all-results(task-t)=null
        and not-have-all-results(task-t)=null
        and AND decompositions
then
        check for all results
        if check shows I have values for all results
        then I have all results
        else if I don't have all results and they are not available
        then I don't have all results
        else if I don't have all results but they are available
        then
                fetch results from all subtasks
                        intend to query my child agents who I believe know these results or intend to
                        compute it myself (if I 'm responsible for subtask)
                if I am proactive or if there is open query from my child agent about resources
                then intend to inform about resources my child agent who I believe desires it


if      agent(self, task-t)
        and able(self, task-t)
```

```
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and have-all-results(task-t)=null
        and not-have-all-results(task-t)=null
        and OR decompositions
then
        check for all results
        if check shows I have values for all results
        then I have all results
        else if I don't have all results and they are not available (all subtasks failed)
        then I don't have all results
        else if I don't have all results but they are available
        then
                fetch results from a subtask
                        intend to query my child agent who I believe know these results or intend to
                        compute it myself (if I 'm responsible for subtask)
                if I am proactive or if there is open query from my child agent about resources
                then intend to inform about resources my child agent who I believe desires it


if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and have-all-results(task-t)=null
        and not-have-all-results(task-t)=null
        and task-t is (executable) method
then
        compute results
                execute method
                if execution succeeded
                then I have all results
                else I don't have all results
```

Supporting operator S5
Task post-processing

```
if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and (status(task-t)=succeeded or status(task-t)=failed)
then
        if status(task-t)=succeeded
        then
                if other agent desires results
                then
                        if I am proactive or if there is open query from agent about results
                        then intend to inform about results agent who I believe desires it

        if status(task-t)=failed
        then
                if other agent desires results
```

then

if I am proactive or if there is open query from agent about results
then intend to inform about status agent who I believe desires it

if any subtask is still relevant and if other agent responsible for subtask
then intend to disconfirm relevancy of my subtask to my child agent

Supporting operators - no conversational triggers

Supporting operator L2
Determine if preconditions are satisfied or not

```
if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=not-attempted
        and have-all-resources(task-t)
        and valid-all-resources(task-t)=null
        and not-valid-all-resources(task-t)=null
then    preconditions_satisfied=check-preconditions(task-t)
                if preconditions_satisfied=true
                then update-beliefs{
                        valid-all-resources(task-t)
                }
                if preconditions_satisfied=false
                then update-beliefs{
                        not-valid-all-resources(task-t)
                }
```

Supporting operator L4
Determine if postconditions are satisfied or not

```
if      agent(self, task-t)
        and able(self, task-t)
        and relevant(task-t)
        and intends(self, task-t)
        and status(task-t)=possible
        and have-all-results(task-t)
        and valid-all-results(task-t)=null
        and not-valid-all-results(task-t)=null
then    postconditions_satisfied=check-postconditions(task-t)
                if postconditions_satisfied=true
                then update-beliefs{
                        valid-all-results(task-t)
                }
                if postconditions_satisfied=false
                then update-beliefs{
                        not-valid-all-results(task-t)
                }
```

Appendix D: Message Generation Routines

Creating conversation intentions

intending-query-ref (general form)

if desire(self, know (self, r, t))
and query-ref produces know(self, r, t) as its rational effect
and there is no such communicational intention
and comm-path(a1)=true
then intend query-ref

intending-inform (general form)

if know(self, desire(a1, know(a1, r, t))
and desire(self, know(a1, r, t))*1 RT (implicit)
and inform produces know(a1, r, t)
and there is no such communicational intention
and comm-path(a1)=true
then intend inform

*1 Comment: I desire to inform you because either it is part of the conversation protocol or I am proactive and helpful agent

RT: real time inference, not an explicit belief

intending-disconfirm

if desire(self, know(a1, p1)*2 RT (implicit)
and disconfirm will produce know(a1, p1)
there is no such communicational intention
and comm-path(a1)=true
then intend disconfirm

*2 Comment:
Disconfirm differs from inform by telling a proposition p1 for which agent believes the other agent believes ~p1. Important implication is that even if the proposition is important for the other agent it does not "desire" it because it already knows ~p1. It is up to the sending agent to understand the importance of informing the other agent even though the other agent does not desire it explicitly. In our case this importance is recognized through the dependencies between tasks.

Formulating Speech Acts

if intend(**query-ref-resource**) with goal to know-resource (self, resource1, task1) =true
and know(self, know-resource(a1, resource1, task1))=true RT
and know-resource(self, resource1, task1)=false
and know(self, intend(a1, inform(self, resource1)=false (implicit)
then formulate SA: query-ref(a1, content: resource, resource1, task1)
and open conversation structures, and keep track of the intended state
associated with this SA

if intend(**query-ref-results**) with goal to know (self, result1, task1)=true
and know(self, knows-result(a1, result, task1))=true RT

and knows-result(self, result1, task1)=false
and know(self, intend(a1, inform(self, result1)=false (implicit)
then formulate the SA: query-ref(a1, content: result, result1, task1)
and open conversation structures, and keep track of the intended state
associated with this SA

if intend(**inform-resource**) with goal to knows-resource(a1, resource1, task1)=true
and knows-resource(self, resource1, task1)=true
and knows-resource(a1, resource1, task1)=false
then formulate SA: inform(a1, content: resource, resource1= r-value, task1)
and open conversation structures, and keep track of the intended state
associated with this SA

if intend(**inform-result**) with goal to know-result(a1, result1, task1)=true
and knows-result(self, result1, task1)=true
and knows-result(a1, result1, task1)=false
then formulate SA: inform(a1, content: result, result1=r-value, task1)
and open conversation structures, and keep track of the intended state
associated with this SA

if intend(**inform-status**) with goal to know-result(a1, result1, task1)=true
and knows-result(self, result1, task1)=true
and knows-result(a1, result1, task1)=false
then formulate SA: inform(a1, content: status, status=failed, task1)
and open conversation structures, and keep track of the intended state
associated with this SA

if intend(**disconfirm-relevancy**) with goal to know(a1, ~relevant(task1)) =true
and know(self, ~relevant(task1))=true (implicit)
and know(self, know(a1, relevant(task1)))=true
then formulate SA: disconfirm(a1, content: relevancy, relevant, task1)
and open conversation structures, and keep track of the intended state
associated with this SA

Appendix E: Conversation traces

(Note: *s* means *sender*, *r* means *receiver*, *c* means *content*, *cid* means *conversation-id)*

Test scenario #1
This conversation trace shows in detail how messages influence beliefs of the senders and receivers.

Messages:
1)
(query-ref :s ag1 :r ag3 :c result air_line T022 :cid 1001)

- a1 new beliefs: [relevant, T022], [intend, ag3, T022], [desire-resource, ag3, destination, T022]
- ag3 new beliefs: [relevant, T011], [intend, ag1, T011], [desire-result, ag1, air_line, T011], [relevant, T022], [intend, ag3, T022], [desire-resource, ag3, destination, T022]

2)
(query-ref :s ag1 :r ag2 :content result destination T021 :cid 1002)

- a1 new beliefs: [relevant, T021], [intend, ag2, T021]
- a2 new beliefs: [relevant, T011], [intend, ag1, T011], [desire-result, ag1, destination, T011], [relevant, T021] , [intend, ag2, T021]

3)
(query-ref :s ag3 :r ag1 :c resource destination T022 :cid 3001)
4)
(inform :s ag2 :r ag1 :c result destination Berlin T021 :cid 1002)

- ag2 new beliefs: [know-result, ag1, destination, T021], [done, T021]
- ag1 new beliefs: [know-result, ag2, destination, T021], [have-result, ag1, destination, T011], [done, T021]

5)
(inform :s ag1 :r ag3 :c resource destination Berlin T022 :cid 3001)

- ag1 new beliefs: [know-resource, ag3, destination, T022]
- ag3 new beliefs: [have-resource, ag3, destination, T022]

6)
(query-ref :s ag3 :r ag2 :c result air_line M022 :cid 3002)

- ag3 new beliefs: [relevant, M022], [intend, ag2, M022]
- ag2 new beliefs: [relevant, T022], [intend, ag3, T022], [desire-result, ag3, air_line, T022], [relevant, M022], [intend, ag2 M022]

7)
(inform :s ag2 :r ag3 :c result air_line Air Canada M022 :cid 3002)

- ag2 new beliefs: [know-result, ag3, air_line, M022], [done, M022]
- ag3 new beliefs: [know-result, ag2, air_line, M022], [done, M022], [have-result, ag3, air_line, T022]

8)
(inform :s ag3 :r ag1 :c result air_line Air Canada T022 :cid 1001)

- ag3 new beliefs: [know-result, ag1, air_line, T022], [done, T022]
- ag1 new beliefs: [know-result, ag3, air_line, T022], [done, T022], [have-result, ag1, air_line, T011]

At the end, ag1 has the following beliefs: [have-all-results, T011], [valid-all-results, T011], [done, T011]. This means that ag1 successfully finalized the top task in the task structure and the performance of the task structure was successful.

Test scenario #2

This conversation trace shows how messages influence relevant beliefs of the senders and receivers.

Messages:

1)

(query-ref :s ag3 :r ag1 :c resource destination T022 :cid 3001)

- ag1 new beliefs: [relevant, T022], [intend, ag3, T022], [desire-resource, ag3, destination, T022]

2)

(query-ref :s ag1 :r ag3 :c result air_line T022 :cid 1002)

- ag3 new beliefs: [desire-result, ag1, air_line, T011]

3)

(query-ref :s ag1 :r ag2 :c result destination T021 :cid 1003)

- ag1 new beliefs: [relevant, T021], [intend, ag2, T021]
- ag2 new beliefs: [desire-result, ag1, destination, T011]

4)

(inform :s ag2 :r ag1 :c result destination Berlin T021 :cid 1003)

- ag2 new beliefs: [know-result, ag1, destination, T021], [done, T021]
- ag1 new beliefs: [know-result, ag2, destination, T021], [have-result, ag1, destination, T011], [done, T021]

5)

(inform :s ag1 :r ag3 :c resource destination Berlin T022 :cid 3001)

- ag1 new beliefs: [know-resource, ag3, destination, T022]
- ag3 new beliefs: [have-resource, ag3, destination, T022]

6)

(query-ref :s ag3 :r ag4 :c result air_line M022 :cid 3003)

- ag3 new beliefs: [relevant, M022], [intend, ag4, M022]
- ag4 new beliefs: [relevant, T022], [intend, ag3, T022], [desire-result, ag3, air_line, T022], [relevant, M022], [intend, ag4 M022]

7)

(inform :s ag3 :r ag4 :c resource destination Berlin M022 :cid 3004)

- ag3 new beliefs: [know-resource, ag4, destination, M022]
- ag4 new beliefs: [have-resource, ag4, destination, M022]

8)

(inform :s ag4 :r ag3 :c result air_line Air Canada M022 :cid 3003)

- ag4 new beliefs: [know-result, ag3, air_line, M022], [done, M022]
- ag3 new beliefs: [know-result, ag4, air_line, M022], [done, M022], [have-result, ag3, air_line, T022]

9)

(inform :s ag3 :r ag1 :c result air_line Air Canada T022 :cid 1002)

- ag3 new beliefs: [know-result, ag1, air_line, T022], [done, T022]
- ag1 new beliefs: [know-result, ag3, air_line, T022], [done, T022], [have-result, ag1, air_line, T011]

At the end, ag1 has the following beliefs: `[have-all-results, T011], [valid-all-results, T011], [done, T011]`. This means that ag1 successfully finalized the top task in the task structure and the performance of the task structure was successful.

Test scenario #3
This conversation trace shows how messages influence relevant beliefs of the senders and receivers.

Messages:
1)
(query-ref :s ag3 :r ag1 :c resource destination T022 :cid 3001)
- `ag1 new beliefs: [relevant, T022], [intend, ag3, T022], [desire-resource, ag3, destination, T022]`

2)
(query-ref :s [redacted] :c result air_line T022 :cid 1002)
- `ag[redacted]eliefs: [desire-result, ag1, air_line, T011]`

3)
(query-ref :s ag1 :r ag2 :c result destination T021 :cid 1003)
- `ag1 new beliefs: [relevant, T021], [intend, ag2, T021]`
- `ag2 new beliefs: [desire-result, ag1, destination, T011]`

4)
(inform :s ag2 :r ag1 :c status failed T021 :cid 1003)
- `ag2 new beliefs: [not-have-result, ag2, destination, T021], [done, T021], [status, failed, T021]`
- `ag1 new beliefs: [done, T021], [status, failed, T021], [not-have-result, ag1, destination, T011], [not-have-all-results, ag1, T011]`

5)
(disconfirm :s ag1 :r ag3 :c relevancy relevant T022 :cid 1004)
- `ag1 new beliefs: NOT[relevant, T022]`
- `ag3 new beliefs: NOT[relevant, T022]`

At the end, ag1 has the following beliefs: `[done, T011], [status, failed, T011]`. This means that ag1 finalized the top task in the task structure with the *failed* status and the task structure execution failed.

Test scenario #4
This conversation trace shows how messages influence relevant beliefs of the senders and receivers.

Messages:
1)
(query-ref :s ag1 :r ag2 :c result destination T021 :cid 1001)
- `a1 new beliefs: [relevant, T021], [intend, ag2, T021]`
- `a2 new beliefs: [relevant, T011], [intend, ag1, T011], [desire-result, ag1, destination, T011], [relevant, T021] , [intend, ag2, T021]`

2)
(inform :s ag2 :r ag1 :c result destination Berlin T021 :cid 1001)

- ag2 new beliefs: [know-result, ag1, destination, T021], [done, T021]
- ag1 new beliefs: [know-result, ag2, destination, T021], [have-result, ag1, destination, T011], [done, T021]

At the end, ag1 has the following beliefs: [have-all-results, T011], [valid-all-results, T011], [done, T011]. This means that ag1 successfully finalized the top task in the task structure and the performance of the task structure was successful.

Test scenario #5

This conversation trace shows how messages influence relevant beliefs of the senders and receivers.

Messages:

1)

(query-ref :s ag1 :r ag2 :c result destination T021 :cid 1001)

- ag1 new beliefs: [relevant, T021], [intend, ag2, T021]
- ag2 new beliefs: [relevant, T011], [intend, ag1, T011], [desire-result, ag1, destination, T011], [relevant, T021] , [intend, ag2, T021]

2)

(inform :s ag2 :r ag1 :c status failed T021 :cid 1001)

- ag2 new beliefs: [not-have-result, ag2, destination, T021], [done, T021], [status, failed, T021]
- ag1 new beliefs: [done, T021], [status, failed, T021], [not-have-result, ag1, destination, T011]

3)

(query-ref :s ag1 :r ag3 :c result destination T022 :cid 1002)

- ag1 new beliefs: [relevant, T022], [intend, ag3, T022]
- ag3 new beliefs: [relevant, T011], [intend, ag1, T011], [desire-result, ag1, destination, T011], [relevant, T022], [intend, ag3, T022]

4)

(inform :s ag3 :r ag1 :c result destination Berlin T022 :cid 1002)

- ag3 new beliefs: [know-result, ag1, destination, T022], [done, T022]
- ag1 new beliefs: [know-result, ag3, destination, T022], [have-result, ag1, destination, T011], [done, T022]

At the end, ag1 has the following beliefs: [have-all-results, T011], [valid-all-results, T011], [done, T011]. This means that ag1 successfully finalized the top task in the task structure and the performance of the task structure was successful.

Test Scenario #6

1)(query-ref :s ag4 :r ag1 :c resource origin T022 :cid 4001)

2)(query-ref :s ag4 :r ag1 :c resource destination T022 :cid 4002)

3)(query-ref :s ag1 :r ag4 :c result airline T022 :cid 1003)

4)(query-ref :s ag1 :r ag3 :c result origin T021 :cid 1004)

5)(query-ref :s ag1 :r ag3 :c result destination T021 :cid 1005)

6)(query-ref :s ag3 :r ag2 :c result origin T031 :cid 3003)

7)(query-ref :s ag3 :r ag2 :c result destination M032 :cid 3004)

8)(inform :s ag2 :r ag3 :c result destination Berlin M032 :cid 3004)

9)(inform :s ag2 :r ag3 :c result origin Edmonton T031 :cid 3003)

10)(inform :s ag3 :r ag1 :c result origin Edmonton T021 :cid 1004)

11)(inform :s ag3 :r ag1 :c result destination Berlin T021 :cid 1005)

12)(inform :s ag1 :r ag4 :c resource origin Edmonton T022 :cid 4001)

13)(inform :s ag1 :r ag4 :c resource destination Berlin T022 :cid 4002)

14)(query-ref :s ag4 :r ag2 :c result airline T033 :cid 4004)

15)(inform :s ag4 :r ag2 :c resource origin Edmonton T033 :cid 4005)

16)(inform :s ag4 :r ag2 :c resource destination Berlin T033 :cid 4006)

17)(inform :s ag2 :r ag4 :c result airline Air Canada T033 :cid 4004)

18)(inform :s ag4 :r ag1 :c result airline Air Canada T022 :c-id 1003)

Test Scenario #7

1)(query-ref :s ag4 :r ag1 :c resource origin T022 :cid 4001)

2)(query-ref :s ag4 :r ag1 :c resource destination T022 :cid 4002)

3)(query-ref :s ag1 :r ag4 :c result airline T022 :cid 1003)

4)(query-ref :s ag1 :r ag3 :c result origin T021 :cid 1004)

5)(query-ref :s ag1 :r ag3 :c result destination T021 :cid 1005)

6)(query-ref :s ag3 :r ag2 :c result origin T031 :cid 3003)

7)(query-ref :s ag3 :r ag2 :c result destination M032 :cid 3004)

8)(inform :s ag2 :r ag3 :c result destination Berlin M032 :cid 3004)

9)(inform :s ag2 :r ag3 :c status failed T031 :cid 3003)

10)(inform :s ag3 :r ag1 :c status failed T021 :cid 1004)

11)(disconfirm :s ag1 :r ag4 :c relevancy relevant T022 :cid 1006)